VectSharp

2.6.1

Generated by Doxygen 1.9.5

1 VectSharp: a light library for C# vector graphics 1
1.1 Introduction . . . 1
1.2 Installing VectSharp . . . 2
1.3 Usage . . . 2
1.4 Font libraries . . . 3
1.5 Creating new output layers . . . 4
1.6 Compiling VectSharp from source . . . 4
1.6.1 Windows . . . 5
1.6.2 macOS and Linux . . . 5
1.7 Note about VectSharp.MuPDFUtils and .NET Framework . . . 5

2 Namespace Index 7
2.1 Package List . . . 7

3 Hierarchical Index 9
3.1 Class Hierarchy . . . 9

4 Class Index 13
4.1 Class List . . . 13

5 File Index 21
5.1 File List . . . 21

6 Namespace Documentation 23
6.1 VectSharp Namespace Reference . . . 23
6.1.1 Enumeration Type Documentation . . . 25
6.1.1.1 FillRule . . . 25
6.1.1.2 LineCaps . . . 26
6.1.1.3 LineJoins . . . 26
6.1.1.4 PixelFormats . . . 26
6.1.1.5 Script . . . 26
6.1.1.6 SegmentType . . . 26
6.1.1.7 TextAnchors . . . 27
6.1.1.8 TextBaselines . . . 27
6.1.1.9 UnbalancedStackActions . . . 27
6.2 VectSharp.Canvas Namespace Reference . . . 27
6.3 VectSharp.Filters Namespace Reference . . . 28
6.4 VectSharp.Fonts Namespace Reference . . . 28
6.5 VectSharp.ImageSharpUtils Namespace Reference . . . 28
6.6 VectSharp.Markdown Namespace Reference . . . 29
6.7 VectSharp.MarkdownCanvas Namespace Reference . . . 29
6.8 VectSharp.MuPDFUtils Namespace Reference . . . 30
6.9 VectSharp.PDF Namespace Reference . . . 30
6.10 VectSharp.PDF.Figures Namespace Reference . . . 30

6.11 VectSharp.PDF.OptionalContentGroups Namespace Reference . . . 30
6.11.1 Enumeration Type Documentation . . . 31
6.11.1.1 OptionalContentGroupBooleanOperation . . . 31
6.12 VectSharp.Plots Namespace Reference . . . 31
6.13 VectSharp.Raster Namespace Reference . . . 33
6.14 VectSharp.Raster.ImageSharp Namespace Reference . . . 33
6.14.1 Enumeration Type Documentation . . . 34
6.14.1.1 OutputFormats . . . 34
6.15 VectSharp.SVG Namespace Reference . . . 34
6.16 VectSharp.ThreeD Namespace Reference . . . 34

7 Class Documentation **35**

7.1 VectSharp.Plots.ActionDataPointElement Class Reference . . . 35
7.1.1 Detailed Description . . . 36
7.1.2 Constructor & Destructor Documentation . . . 36
7.1.2.1 ActionDataPointElement() . . . 36
7.1.3 Member Function Documentation . . . 36
7.1.3.1 Plot() . . . 36
7.1.4 Property Documentation . . . 37
7.1.4.1 PlotAction . . . 37
7.2 VectSharp.ThreeD.AmbientLightSource Class Reference . . . 37
7.2.1 Detailed Description . . . 38
7.2.2 Constructor & Destructor Documentation . . . 38
7.2.2.1 AmbientLightSource() . . . 38
7.2.3 Member Function Documentation . . . 39
7.2.3.1 GetLightAt() . . . 39
7.2.3.2 GetObstruction() . . . 39
7.2.4 Property Documentation . . . 39
7.2.4.1 CastsShadow . . . 40
7.2.4.2 Intensity . . . 40
7.3 VectSharp.Canvas.AnimatedCanvas Class Reference . . . 40
7.3.1 Detailed Description . . . 41
7.3.2 Member Function Documentation . . . 41
7.3.2.1 Render() . . . 41
7.3.3 Member Data Documentation . . . 41
7.3.3.1 CurrentFrameProperty . . . 42
7.3.3.2 FrameCountProperty . . . 42
7.3.3.3 FrameRateProperty . . . 42
7.3.3.4 IsPlayingProperty . . . 43
7.3.4 Property Documentation . . . 43
7.3.4.1 CurrentFrame . . . 43
7.3.4.2 FrameCount . . . 43

7.3.4.3 FrameRate . . . 43
7.3.4.4 IsPlaying . . . 44
7.4 VectSharp.AnimatedPNG Class Reference . . . 44
7.4.1 Detailed Description . . . 44
7.4.2 Member Enumeration Documentation . . . 44
7.4.2.1 InterframeCompression . . . 45
7.4.3 Member Function Documentation . . . 45
7.4.3.1 Create() . . . 45
7.5 VectSharp.Animation Class Reference . . . 45
7.5.1 Detailed Description . . . 46
7.5.2 Constructor & Destructor Documentation . . . 46
7.5.2.1 Animation() . . . 46
7.5.3 Member Function Documentation . . . 47
7.5.3.1 AddFrame() . . . 47
7.5.3.2 GetFrameAtAbsolute() . . . 47
7.5.3.3 GetFrameAtRelative() . . . 48
7.5.3.4 RemoveLastFrame() . . . 48
7.5.4 Property Documentation . . . 48
7.5.4.1 Background . . . 48
7.5.4.2 Duration . . . 49
7.5.4.3 Frames . . . 49
7.5.4.4 Height . . . 49
7.5.4.5 LinearisationResolution . . . 49
7.5.4.6 RepeatCount . . . 49
7.5.4.7 Transitions . . . 50
7.5.4.8 Width . . . 50
7.6 VectSharp.PDF.AnnotationStyle Class Reference . . . 50
7.6.1 Detailed Description . . . 50
7.6.2 Constructor & Destructor Documentation . . . 51
7.6.2.1 AnnotationStyle() . . . 51
7.6.3 Property Documentation . . . 51
7.6.3.1 BorderColour . . . 51
7.6.3.2 BorderDash . . . 51
7.6.3.3 BorderWidth . . . 51
7.7 VectSharp.PDF.AnnotationStyleCollection Class Reference . . . 52
7.7.1 Detailed Description . . . 53
7.7.2 Constructor & Destructor Documentation . . . 53
7.7.2.1 AnnotationStyleCollection() . . . 53
7.7.3 Member Function Documentation . . . 53
7.7.3.1 Add() [1/2] . . . 53
7.7.3.2 Add() [2/2] . . . 53
7.7.3.3 Clear() . . . 54

7.7.3.4 Contains() . . . 54
7.7.3.5 ContainsKey() . . . 54
7.7.3.6 CopyTo() . . . 54
7.7.3.7 GetEnumerator() . . . 54
7.7.3.8 Remove() [1/2] . . . 54
7.7.3.9 Remove() [2/2] . . . 55
7.7.3.10 TryGetValue() . . . 55
7.7.4 Property Documentation . . . 55
7.7.4.1 Count . . . 55
7.7.4.2 DefaultStyle . . . 55
7.7.4.3 IsReadOnly . . . 55
7.7.4.4 Keys . . . 56
7.7.4.5 Styles . . . 56
7.7.4.6 this[string key] . . . 56
7.7.4.7 Values . . . 56
7.8 VectSharp.Plots.Area< T > Class Template Reference . . . 56
7.8.1 Detailed Description . . . 57
7.8.2 Constructor & Destructor Documentation . . . 57
7.8.2.1 Area() . . . 57
7.8.3 Member Function Documentation . . . 59
7.8.3.1 Plot() . . . 59
7.8.4 Property Documentation . . . 59
7.8.4.1 CoordinateSystem . . . 59
7.8.4.2 Data . . . 60
7.8.4.3 GetBaseline . . . 60
7.8.4.4 PresentationAttributes . . . 60
7.8.4.5 Smooth . . . 60
7.8.4.6 Tag . . . 60
7.9 VectSharp.ThreeD.AreaLightSource Class Reference . . . 61
7.9.1 Detailed Description . . . 62
7.9.2 Constructor & Destructor Documentation . . . 62
7.9.2.1 AreaLightSource() . . . 62
7.9.3 Member Function Documentation . . . 63
7.9.3.1 GetLightAt() . . . 63
7.9.3.2 GetObstruction() . . . 63
7.9.4 Property Documentation . . . 64
7.9.4.1 CastsShadow . . . 64
7.9.4.2 Center . . . 64
7.9.4.3 Direction . . . 64
7.9.4.4 DistanceAttenuationExponent . . . 64
7.9.4.5 Intensity . . . 65
7.9.4.6 PenumbraAttenuationExponent . . . 65

7.9.4.7 PenumbraRadius . . . 65
7.9.4.8 Radius . . . 65
7.9.4.9 ShadowSamplingPointCount . . . 65
7.9.4.10 SourceDistance . . . 66
7.10 VectSharp.Canvas.AvaloniaContextInterpreter Class Reference . . . 66
7.10.1 Detailed Description . . . 66
7.10.2 Member Enumeration Documentation . . . 67
7.10.2.1 TextOptions . . . 67
7.10.3 Member Function Documentation . . . 67
7.10.3.1 PaintToAnimatedCanvas() . . . 67
7.10.3.2 PaintToCanvas() [1/4] . . . 67
7.10.3.3 PaintToCanvas() [2/4] . . . 68
7.10.3.4 PaintToCanvas() [3/4] . . . 69
7.10.3.5 PaintToCanvas() [4/4] . . . 69
7.11 VectSharp.Plots.Bars< T > Class Template Reference . . . 70
7.11.1 Detailed Description . . . 71
7.11.2 Constructor & Destructor Documentation . . . 71
7.11.2.1 Bars() [1/6] . . . 72
7.11.2.2 Bars() [2/6] . . . 72
7.11.2.3 Bars() [3/6] . . . 72
7.11.2.4 Bars() [4/6] . . . 73
7.11.2.5 Bars() [5/6] . . . 73
7.11.2.6 Bars() [6/6] . . . 74
7.11.3 Member Function Documentation . . . 74
7.11.3.1 Plot() . . . 74
7.11.4 Property Documentation . . . 75
7.11.4.1 CoordinateSystem . . . 75
7.11.4.2 Data . . . 75
7.11.4.3 GetBaseline . . . 75
7.11.4.4 Margin . . . 75
7.11.4.5 PresentationAttributes . . . 76
7.11.4.6 Tag . . . 76
7.12 VectSharp.TrueTypeFile.Bearings Struct Reference . . . 76
7.12.1 Detailed Description . . . 76
7.12.2 Member Data Documentation . . . 76
7.12.2.1 LeftSideBearing . . . 77
7.12.2.2 RightSideBearing . . . 77
7.13 VectSharp.Markdown.BlockRenderedEventArgs Class Reference . . . 77
7.13.1 Detailed Description . . . 78
7.13.2 Constructor & Destructor Documentation . . . 78
7.13.2.1 BlockRenderedEventArgs() . . . 78
7.13.3 Property Documentation . . . 78

7.13.3.1 Block . . . 78
7.13.3.2 Graphics . . . 79
7.13.3.3 MarkdownContext . . . 79
7.14 VectSharp.Markdown.BlockRenderingEventArgs Class Reference . . . 79
7.14.1 Detailed Description . . . 80
7.14.2 Constructor & Destructor Documentation . . . 80
7.14.2.1 BlockRenderingEventArgs() . . . 80
7.14.3 Property Documentation . . . 80
7.14.3.1 Block . . . 80
7.14.3.2 Graphics . . . 81
7.14.3.3 MarkdownContext . . . 81
7.15 VectSharp.Filters.BoxBlurFilter Class Reference . . . 81
7.15.1 Detailed Description . . . 82
7.15.2 Constructor & Destructor Documentation . . . 82
7.15.2.1 BoxBlurFilter() . . . 82
7.15.3 Member Function Documentation . . . 82
7.15.3.1 Filter() . . . 83
7.15.4 Property Documentation . . . 83
7.15.4.1 BottomRightMargin . . . 83
7.15.4.2 BoxRadius . . . 83
7.15.4.3 TopLeftMargin . . . 84
7.16 VectSharp.Plots.BoxPlot Class Reference . . . 84
7.16.1 Detailed Description . . . 85
7.16.2 Constructor & Destructor Documentation . . . 85
7.16.2.1 BoxPlot() . . . 86
7.16.3 Member Function Documentation . . . 86
7.16.3.1 Plot() . . . 86
7.16.4 Property Documentation . . . 87
7.16.4.1 Box1 . . . 87
7.16.4.2 Box2 . . . 87
7.16.4.3 BoxPresentationAttributes . . . 87
7.16.4.4 CentreSymbol . . . 87
7.16.4.5 CentreSymbolPresentationAttributes . . . 88
7.16.4.6 CoordinateSystem . . . 88
7.16.4.7 Direction . . . 88
7.16.4.8 NotchSize . . . 88
7.16.4.9 NotchWidth . . . 88
7.16.4.10 Position . . . 89
7.16.4.11 Tag . . . 89
7.16.4.12 Whisker1 . . . 89
7.16.4.13 Whisker2 . . . 89
7.16.4.14 WhiskersPresentationAttributes . . . 89

7.16.4.15 WhiskerWidth . . . 90
7.16.4.16 Width . . . 90
7.17 VectSharp.Brush Class Reference . . . 90
7.17.1 Detailed Description . . . 91
7.17.2 Member Function Documentation . . . 91
7.17.2.1 MultiplyOpacity() . . . 91
7.17.2.2 operator Brush() . . . 91
7.18 VectSharp.Plots.CategoricalBars< T > Class Template Reference . . . 92
7.18.1 Detailed Description . . . 92
7.18.2 Constructor & Destructor Documentation . . . 93
7.18.2.1 CategoricalBars() [1/4] . . . 93
7.18.2.2 CategoricalBars() [2/4] . . . 93
7.18.2.3 CategoricalBars() [3/4] . . . 94
7.18.2.4 CategoricalBars() [4/4] . . . 94
7.19 VectSharp.Plots.CategoricalCoordinateSystem1D< T > Class Template Reference . . . 94
7.19.1 Detailed Description . . . 95
7.19.2 Constructor & Destructor Documentation . . . 96
7.19.2.1 CategoricalCoordinateSystem1D() [1/2] . . . 96
7.19.2.2 CategoricalCoordinateSystem1D() [2/2] . . . 96
7.19.3 Member Function Documentation . . . 96
7.19.3.1 ToPlotCoordinates() . . . 96
7.19.4 Property Documentation . . . 97
7.19.4.1 Coordinates . . . 97
7.20 VectSharp.TrueTypeFile.ClassDefinitionTable.ClassRangeRecord Struct Reference . . . 97
7.20.1 Detailed Description . . . 97
7.20.2 Constructor & Destructor Documentation . . . 98
7.20.2.1 ClassRangeRecord() . . . 98
7.20.3 Member Data Documentation . . . 98
7.20.3.1 Class . . . 98
7.20.3.2 EndGlyphID . . . 98
7.20.3.3 StartGlyphID . . . 98
7.21 VectSharp.Plots.ClusteredBars Class Reference . . . 99
7.21.1 Detailed Description . . . 100
7.21.2 Constructor & Destructor Documentation . . . 100
7.21.2.1 ClusteredBars() [1/3] . . . 100
7.21.2.2 ClusteredBars() [2/3] . . . 101
7.21.2.3 ClusteredBars() [3/3] . . . 101
7.21.3 Member Function Documentation . . . 102
7.21.3.1 Plot() . . . 102
7.21.4 Property Documentation . . . 102
7.21.4.1 CoordinateSystem . . . 102
7.21.4.2 Data . . . 102

7.21.4.3 GetBaseline . . . 103
7.21.4.4 InterClusterMargin . . . 103
7.21.4.5 IntraClusterMargin . . . 103
7.21.4.6 PresentationAttributes . . . 103
7.21.4.7 Tag . . . 103
7.21.4.8 Vertical . . . 104
7.22 VectSharp.Colour Struct Reference . . . 104
7.22.1 Detailed Description . . . 106
7.22.2 Member Function Documentation . . . 106
7.22.2.1 Equals() [1/2] . . . 106
7.22.2.2 Equals() [2/2] . . . 106
7.22.2.3 FromCSSString() . . . 106
7.22.2.4 FromHSL() . . . 107
7.22.2.5 FromLab() . . . 107
7.22.2.6 FromRgb() [1/3] . . . 107
7.22.2.7 FromRgb() [2/3] . . . 108
7.22.2.8 FromRgb() [3/3] . . . 108
7.22.2.9 FromRgba() [1/6] . . . 109
7.22.2.10 FromRgba() [2/6] . . . 109
7.22.2.11 FromRgba() [3/6] . . . 110
7.22.2.12 FromRgba() [4/6] . . . 110
7.22.2.13 FromRgba() [5/6] . . . 111
7.22.2.14 FromRgba() [6/6] . . . 111
7.22.2.15 FromXYZ() . . . 112
7.22.2.16 GetHashCode() . . . 112
7.22.2.17 operator"!=() . . . 112
7.22.2.18 operator==() . . . 113
7.22.2.19 ToCSSString() . . . 113
7.22.2.20 ToHSL() . . . 113
7.22.2.21 ToLab() . . . 113
7.22.2.22 ToXYZ() . . . 114
7.22.2.23 WithAlpha() [1/4] . . . 114
7.22.2.24 WithAlpha() [2/4] . . . 114
7.22.2.25 WithAlpha() [3/4] . . . 115
7.22.2.26 WithAlpha() [4/4] . . . 115
7.22.3 Member Data Documentation . . . 115
7.22.3.1 A . . . 115
7.22.3.2 a . . . 116
7.22.3.3 B . . . 116
7.22.3.4 G . . . 116
7.22.3.5 H . . . 116
7.22.3.6 L . . . 116

7.22.3.7 R . . . 117
7.22.3.8 S . . . 117
7.22.3.9 X . . . 117
7.22.3.10 Y . . . 117
7.23 VectSharp.ThreeD.ColourMaterial Class Reference . . . 118
7.23.1 Detailed Description . . . 118
7.23.2 Constructor & Destructor Documentation . . . 119
7.23.2.1 ColourMaterial() . . . 119
7.23.3 Member Function Documentation . . . 119
7.23.3.1 GetColour() . . . 119
7.23.4 Property Documentation . . . 120
7.23.4.1 Colour . . . 120
7.24 VectSharp.Filters.ColourMatrix Class Reference . . . 120
7.24.1 Detailed Description . . . 122
7.24.2 Constructor & Destructor Documentation . . . 122
7.24.2.1 ColourMatrix() . . . 122
7.24.3 Member Function Documentation . . . 122
7.24.3.1 Apply() [1/5] . . . 123
7.24.3.2 Apply() [2/5] . . . 123
7.24.3.3 Apply() [3/5] . . . 124
7.24.3.4 Apply() [4/5] . . . 124
7.24.3.5 Apply() [5/5] . . . 124
7.24.3.6 LuminanceToAlpha() . . . 125
7.24.3.7 LuminanceToColour() . . . 125
7.24.3.8 operator∗() . . . 126
7.24.3.9 ToColour() . . . 126
7.24.3.10 WithAlpha() . . . 127
7.24.4 Member Data Documentation . . . 127
7.24.4.1 AlphaInversion . . . 127
7.24.4.2 GreyScale . . . 127
7.24.4.3 Identity . . . 128
7.24.4.4 Inversion . . . 128
7.24.4.5 InvertedAlphaShift . . . 128
7.24.4.6 Pastel . . . 128
7.24.5 Property Documentation . . . 128
7.24.5.1 A1 . . . 129
7.24.5.2 A2 . . . 129
7.24.5.3 A3 . . . 129
7.24.5.4 A4 . . . 129
7.24.5.5 A5 . . . 129
7.24.5.6 B1 . . . 130
7.24.5.7 B2 . . . 130

7.24.5.8 B3 . . . 130
7.24.5.9 B4 . . . 130
7.24.5.10 B5 . . . 130
7.24.5.11 G1 . . . 131
7.24.5.12 G2 . . . 131
7.24.5.13 G3 . . . 131
7.24.5.14 G4 . . . 131
7.24.5.15 G5 . . . 131
7.24.5.16 R1 . . . 132
7.24.5.17 R2 . . . 132
7.24.5.18 R3 . . . 132
7.24.5.19 R4 . . . 132
7.24.5.20 R5 . . . 132
7.24.5.21 this[int y, int x] . . . 132
7.25 VectSharp.Filters.ColourMatrixFilter Class Reference . . . 133
7.25.1 Detailed Description . . . 134
7.25.2 Constructor & Destructor Documentation . . . 134
7.25.2.1 ColourMatrixFilter() . . . 134
7.25.3 Member Function Documentation . . . 134
7.25.3.1 Filter() . . . 135
7.25.4 Property Documentation . . . 135
7.25.4.1 BottomRightMargin . . . 135
7.25.4.2 ColourMatrix . . . 135
7.25.4.3 TopLeftMargin . . . 136
7.26 VectSharp.Colours Class Reference . . . 136
7.26.1 Detailed Description . . . 142
7.26.2 Member Data Documentation . . . 142
7.26.2.1 AliceBlue . . . 142
7.26.2.2 AntiqueWhite . . . 142
7.26.2.3 Aqua . . . 143
7.26.2.4 Aquamarine . . . 143
7.26.2.5 Azure . . . 143
7.26.2.6 Beige . . . 143
7.26.2.7 Bisque . . . 143
7.26.2.8 Black . . . 144
7.26.2.9 BlanchedAlmond . . . 144
7.26.2.10 Blue . . . 144
7.26.2.11 BlueViolet . . . 144
7.26.2.12 Brown . . . 144
7.26.2.13 BurlyWood . . . 145
7.26.2.14 CadetBlue . . . 145
7.26.2.15 Chartreuse . . . 145

7.26.2.16 Chocolate . . . 145
7.26.2.17 Coral . . . 145
7.26.2.18 CornflowerBlue . . . 146
7.26.2.19 Cornsilk . . . 146
7.26.2.20 Crimson . . . 146
7.26.2.21 Cyan . . . 146
7.26.2.22 DarkBlue . . . 146
7.26.2.23 DarkCyan . . . 147
7.26.2.24 DarkGoldenRod . . . 147
7.26.2.25 DarkGray . . . 147
7.26.2.26 DarkGreen . . . 147
7.26.2.27 DarkGrey . . . 147
7.26.2.28 DarkKhaki . . . 148
7.26.2.29 DarkMagenta . . . 148
7.26.2.30 DarkOliveGreen . . . 148
7.26.2.31 DarkOrange . . . 148
7.26.2.32 DarkOrchid . . . 148
7.26.2.33 DarkRed . . . 149
7.26.2.34 DarkSalmon . . . 149
7.26.2.35 DarkSeaGreen . . . 149
7.26.2.36 DarkSlateBlue . . . 149
7.26.2.37 DarkSlateGray . . . 149
7.26.2.38 DarkSlateGrey . . . 150
7.26.2.39 DarkTurquoise . . . 150
7.26.2.40 DarkViolet . . . 150
7.26.2.41 DeepPink . . . 150
7.26.2.42 DeepSkyBlue . . . 150
7.26.2.43 DimGray . . . 151
7.26.2.44 DimGrey . . . 151
7.26.2.45 DodgerBlue . . . 151
7.26.2.46 FireBrick . . . 151
7.26.2.47 FloralWhite . . . 151
7.26.2.48 ForestGreen . . . 152
7.26.2.49 Fuchsia . . . 152
7.26.2.50 Gainsboro . . . 152
7.26.2.51 GhostWhite . . . 152
7.26.2.52 Gold . . . 152
7.26.2.53 GoldenRod . . . 153
7.26.2.54 Gray . . . 153
7.26.2.55 Green . . . 153
7.26.2.56 GreenYellow . . . 153
7.26.2.57 Grey . . . 153

7.26.2.58 HoneyDew . . . 154
7.26.2.59 HotPink . . . 154
7.26.2.60 IndianRed . . . 154
7.26.2.61 Indigo . . . 154
7.26.2.62 Ivory . . . 154
7.26.2.63 Khaki . . . 155
7.26.2.64 Lavender . . . 155
7.26.2.65 LavenderBlush . . . 155
7.26.2.66 LawnGreen . . . 155
7.26.2.67 LemonChiffon . . . 155
7.26.2.68 LightBlue . . . 156
7.26.2.69 LightCoral . . . 156
7.26.2.70 LightCyan . . . 156
7.26.2.71 LightGoldenRodYellow . . . 156
7.26.2.72 LightGray . . . 156
7.26.2.73 LightGreen . . . 157
7.26.2.74 LightGrey . . . 157
7.26.2.75 LightPink . . . 157
7.26.2.76 LightSalmon . . . 157
7.26.2.77 LightSeaGreen . . . 157
7.26.2.78 LightSkyBlue . . . 158
7.26.2.79 LightSlateGray . . . 158
7.26.2.80 LightSlateGrey . . . 158
7.26.2.81 LightSteelBlue . . . 158
7.26.2.82 LightYellow . . . 158
7.26.2.83 Lime . . . 159
7.26.2.84 LimeGreen . . . 159
7.26.2.85 Linen . . . 159
7.26.2.86 Magenta . . . 159
7.26.2.87 Maroon . . . 159
7.26.2.88 MediumAquaMarine . . . 160
7.26.2.89 MediumBlue . . . 160
7.26.2.90 MediumOrchid . . . 160
7.26.2.91 MediumPurple . . . 160
7.26.2.92 MediumSeaGreen . . . 160
7.26.2.93 MediumSlateBlue . . . 161
7.26.2.94 MediumSpringGreen . . . 161
7.26.2.95 MediumTurquoise . . . 161
7.26.2.96 MediumVioletRed . . . 161
7.26.2.97 MidnightBlue . . . 161
7.26.2.98 MintCream . . . 162
7.26.2.99 MistyRose . . . 162

7.26.2.100 Moccasin . . . 162
7.26.2.101 NavajoWhite . . . 162
7.26.2.102 Navy . . . 162
7.26.2.103 OldLace . . . 163
7.26.2.104 Olive . . . 163
7.26.2.105 OliveDrab . . . 163
7.26.2.106 Orange . . . 163
7.26.2.107 OrangeRed . . . 163
7.26.2.108 Orchid . . . 164
7.26.2.109 PaleGoldenRod . . . 164
7.26.2.110 PaleGreen . . . 164
7.26.2.111 PaleTurquoise . . . 164
7.26.2.112 PaleVioletRed . . . 164
7.26.2.113 PapayaWhip . . . 165
7.26.2.114 PeachPuff . . . 165
7.26.2.115 Peru . . . 165
7.26.2.116 Pink . . . 165
7.26.2.117 Plum . . . 165
7.26.2.118 PowderBlue . . . 166
7.26.2.119 Purple . . . 166
7.26.2.120 RebeccaPurple . . . 166
7.26.2.121 Red . . . 166
7.26.2.122 RosyBrown . . . 166
7.26.2.123 RoyalBlue . . . 167
7.26.2.124 SaddleBrown . . . 167
7.26.2.125 Salmon . . . 167
7.26.2.126 SandyBrown . . . 167
7.26.2.127 SeaGreen . . . 167
7.26.2.128 SeaShell . . . 168
7.26.2.129 Sienna . . . 168
7.26.2.130 Silver . . . 168
7.26.2.131 SkyBlue . . . 168
7.26.2.132 SlateBlue . . . 168
7.26.2.133 SlateGray . . . 169
7.26.2.134 SlateGrey . . . 169
7.26.2.135 Snow . . . 169
7.26.2.136 SpringGreen . . . 169
7.26.2.137 SteelBlue . . . 169
7.26.2.138 Tan . . . 170
7.26.2.139 Teal . . . 170
7.26.2.140 Thistle . . . 170
7.26.2.141 Tomato . . . 170

7.26.2.142 Turquoise . . . 170
7.26.2.143 Violet . . . 171
7.26.2.144 Wheat . . . 171
7.26.2.145 White . . . 171
7.26.2.146 WhiteSmoke . . . 171
7.26.2.147 Yellow . . . 171
7.26.2.148 YellowGreen . . . 172
7.27 VectSharp.Plots.CompositeCoordinateSystem2D< T1, T2 > Class Template Reference . . . 172
7.27.1 Detailed Description . . . 172
7.27.2 Constructor & Destructor Documentation . . . 173
7.27.2.1 CompositeCoordinateSystem2D() . . . 173
7.27.3 Member Function Documentation . . . 173
7.27.3.1 ToPlotCoordinates() . . . 173
7.27.4 Property Documentation . . . 173
7.27.4.1 CoordinateSystemX . . . 174
7.27.4.2 CoordinateSystemY . . . 174
7.28 VectSharp.Filters.CompositeLocationInvariantFilter Class Reference . . . 174
7.28.1 Detailed Description . . . 175
7.28.2 Constructor & Destructor Documentation . . . 175
7.28.2.1 CompositeLocationInvariantFilter() [1/2] . . . 175
7.28.2.2 CompositeLocationInvariantFilter() [2/2] . . . 175
7.28.3 Member Function Documentation . . . 176
7.28.3.1 Filter() . . . 176
7.28.4 Property Documentation . . . 176
7.28.4.1 BottomRightMargin . . . 176
7.28.4.2 Filters . . . 177
7.28.4.3 TopLeftMargin . . . 177
7.29 VectSharp.AnimatedPNG.CompressedFrame Class Reference . . . 177
7.29.1 Detailed Description . . . 178
7.29.2 Constructor & Destructor Documentation . . . 178
7.29.2.1 CompressedFrame() [1/2] . . . 178
7.29.2.2 CompressedFrame() [2/2] . . . 179
7.29.3 Member Function Documentation . . . 179
7.29.3.1 Dispose() . . . 179
7.29.4 Property Documentation . . . 179
7.29.4.1 Duration . . . 179
7.30 VectSharp.Plots.ContinuousAxis Class Reference . . . 180
7.30.1 Detailed Description . . . 181
7.30.2 Constructor & Destructor Documentation . . . 181
7.30.2.1 ContinuousAxis() . . . 181
7.30.3 Member Function Documentation . . . 181
7.30.3.1 Plot() . . . 181

7.30.4 Property Documentation . . . 182
7.30.4.1 ArrowSize . . . 182
7.30.4.2 CoordinateSystem . . . 182
7.30.4.3 EndPoint . . . 182
7.30.4.4 PresentationAttributes . . . 182
7.30.4.5 StartPoint . . . 183
7.30.4.6 Tag . . . 183
7.31 VectSharp.Plots.ContinuousAxisLabels Class Reference . . . 183
7.31.1 Detailed Description . . . 184
7.31.2 Constructor & Destructor Documentation . . . 184
7.31.2.1 ContinuousAxisLabels() . . . 184
7.31.3 Member Function Documentation . . . 185
7.31.3.1 Plot() . . . 185
7.31.4 Property Documentation . . . 185
7.31.4.1 Alignment . . . 185
7.31.4.2 Baseline . . . 185
7.31.4.3 CoordinateSystem . . . 186
7.31.4.4 EndPoint . . . 186
7.31.4.5 IntervalCount . . . 186
7.31.4.6 Position . . . 186
7.31.4.7 PresentationAttributes . . . 186
7.31.4.8 Rotation . . . 187
7.31.4.9 StartPoint . . . 187
7.31.4.10 Tag . . . 187
7.31.4.11 TextFormat . . . 187
7.32 VectSharp.Plots.ContinuousAxisTicks Class Reference . . . 188
7.32.1 Detailed Description . . . 189
7.32.2 Constructor & Destructor Documentation . . . 189
7.32.2.1 ContinuousAxisTicks() . . . 189
7.32.3 Member Function Documentation . . . 189
7.32.3.1 Plot() . . . 189
7.32.4 Property Documentation . . . 190
7.32.4.1 CoordinateSystem . . . 190
7.32.4.2 EndPoint . . . 190
7.32.4.3 IntervalCount . . . 190
7.32.4.4 PresentationAttributes . . . 191
7.32.4.5 SizeAbove . . . 191
7.32.4.6 SizeBelow . . . 191
7.32.4.7 StartPoint . . . 191
7.32.4.8 Tag . . . 192
7.33 VectSharp.Plots.ContinuousAxisTitle Class Reference . . . 192
7.33.1 Detailed Description . . . 193

7.33.2 Constructor & Destructor Documentation . . . 193
7.33.2.1 ContinuousAxisTitle() [1/2] . . . 193
7.33.2.2 ContinuousAxisTitle() [2/2] . . . 194
7.33.3 Member Function Documentation . . . 194
7.33.3.1 Plot() . . . 194
7.33.4 Property Documentation . . . 195
7.33.4.1 Alignment . . . 195
7.33.4.2 Baseline . . . 195
7.33.4.3 CoordinateSystem . . . 195
7.33.4.4 EndPoint . . . 195
7.33.4.5 FollowAxis . . . 196
7.33.4.6 Position . . . 196
7.33.4.7 PresentationAttributes . . . 196
7.33.4.8 Rotation . . . 196
7.33.4.9 StartPoint . . . 196
7.33.4.10 Tag . . . 197
7.33.4.11 Title . . . 197
7.34 VectSharp.Filters.ConvolutionFilter Class Reference . . . 197
7.34.1 Detailed Description . . . 198
7.34.2 Constructor & Destructor Documentation . . . 198
7.34.2.1 ConvolutionFilter() . . . 198
7.34.3 Member Function Documentation . . . 199
7.34.3.1 Filter() . . . 199
7.34.4 Property Documentation . . . 199
7.34.4.1 Bias . . . 200
7.34.4.2 BottomRightMargin . . . 200
7.34.4.3 Kernel . . . 200
7.34.4.4 Normalisation . . . 200
7.34.4.5 PreserveAlpha . . . 200
7.34.4.6 Scale . . . 201
7.34.4.7 TopLeftMargin . . . 201
7.35 VectSharp.Plots.CoordinateSystem< T > Class Template Reference . . . 201
7.35.1 Detailed Description . . . 202
7.35.2 Constructor & Destructor Documentation . . . 202
7.35.2.1 CoordinateSystem() . . . 202
7.35.3 Member Function Documentation . . . 202
7.35.3.1 ToPlotCoordinates() . . . 203
7.35.4 Property Documentation . . . 203
7.35.4.1 CoordinateFunction . . . 203
7.36 VectSharp.Plots.CoordinateSystem1D< T > Class Template Reference . . . 203
7.36.1 Detailed Description . . . 204
7.36.2 Constructor & Destructor Documentation . . . 204

7.36.2.1 CoordinateSystem1D() . . . 204
7.36.3 Member Function Documentation . . . 205
7.36.3.1 ToPlotCoordinates() . . . 205
7.36.4 Property Documentation . . . 205
7.36.4.1 CoordinateFunction . . . 205
7.37 VectSharp.Plots.DataLabels< T > Class Template Reference . . . 206
7.37.1 Detailed Description . . . 207
7.37.2 Constructor & Destructor Documentation . . . 207
7.37.2.1 DataLabels() . . . 207
7.37.3 Member Function Documentation . . . 207
7.37.3.1 Plot() . . . 208
7.37.4 Property Documentation . . . 208
7.37.4.1 Alignment . . . 208
7.37.4.2 Baseline . . . 208
7.37.4.3 CoordinateSystem . . . 208
7.37.4.4 Data . . . 209
7.37.4.5 Label . . . 209
7.37.4.6 Margin . . . 209
7.37.4.7 PresentationAttributes . . . 209
7.37.4.8 Rotation . . . 210
7.37.4.9 Tag . . . 210
7.38 VectSharp.Plots.DataLine< T > Class Template Reference . . . 210
7.38.1 Detailed Description . . . 211
7.38.2 Constructor & Destructor Documentation . . . 211
7.38.2.1 DataLine() . . . 211
7.38.3 Member Function Documentation . . . 212
7.38.3.1 Plot() . . . 212
7.38.4 Property Documentation . . . 212
7.38.4.1 CoordinateSystem . . . 212
7.38.4.2 Data . . . 212
7.38.4.3 PresentationAttributes . . . 213
7.38.4.4 Smooth . . . 213
7.38.4.5 Tag . . . 213
7.39 VectSharp.DefaultFontLibrary Class Reference . . . 213
7.39.1 Detailed Description . . . 214
7.39.2 Member Function Documentation . . . 214
7.39.2.1 ResolveFontFamily() [1/2] . . . 214
7.39.2.2 ResolveFontFamily() [2/2] . . . 215
7.40 VectSharp.Font.DetailedFontMetrics Class Reference . . . 215
7.40.1 Detailed Description . . . 216
7.40.2 Property Documentation . . . 216
7.40.2.1 AdvanceWidth . . . 216

7.40.2.2 Bottom . . . 216
7.40.2.3 Height . . . 216
7.40.2.4 LeftSideBearing . . . 216
7.40.2.5 RightSideBearing . . . 217
7.40.2.6 Top . . . 217
7.40.2.7 Width . . . 217
7.41 VectSharp.DisposableIntPtr Class Reference . . . 217
7.41.1 Detailed Description . . . 218
7.41.2 Constructor & Destructor Documentation . . . 218
7.41.2.1 DisposableIntPtr() . . . 218
7.41.3 Member Function Documentation . . . 218
7.41.3.1 Dispose() . . . 218
7.41.4 Member Data Documentation . . . 219
7.41.4.1 InternalPointer . . . 219
7.42 VectSharp.Document Class Reference . . . 219
7.42.1 Detailed Description . . . 219
7.42.2 Constructor & Destructor Documentation . . . 219
7.42.2.1 Document() . . . 219
7.42.3 Member Data Documentation . . . 220
7.42.3.1 Pages . . . 220
7.43 VectSharp.Plots.ExponentialTrendLine Class Reference . . . 220
7.43.1 Detailed Description . . . 221
7.43.2 Constructor & Destructor Documentation . . . 221
7.43.2.1 ExponentialTrendLine() [1/3] . . . 221
7.43.2.2 ExponentialTrendLine() [2/3] . . . 222
7.43.2.3 ExponentialTrendLine() [3/3] . . . 222
7.43.3 Member Function Documentation . . . 223
7.43.3.1 Plot() . . . 223
7.43.4 Property Documentation . . . 223
7.43.4.1 CoordinateSystem . . . 223
7.43.4.2 Intercept . . . 223
7.43.4.3 MaxX . . . 224
7.43.4.4 MaxY . . . 224
7.43.4.5 MinX . . . 224
7.43.4.6 MinY . . . 224
7.43.4.7 PresentationAttributes . . . 224
7.43.4.8 Slope . . . 225
7.43.4.9 Tag . . . 225
7.44 VectSharp.Canvas.FilterOption Class Reference . . . 225
7.44.1 Detailed Description . . . 226
7.44.2 Member Enumeration Documentation . . . 226
7.44.2.1 FilterOperations . . . 226

7.44.3 Constructor & Destructor Documentation . . . 226
7.44.3.1 FilterOption() . . . 226
7.44.4 Member Data Documentation . . . 227
7.44.4.1 Default . . . 227
7.44.5 Property Documentation . . . 227
7.44.5.1 Operation . . . 227
7.44.5.2 RasterisationResolution . . . 227
7.44.5.3 RasterisationResolutionRelative . . . 228
7.45 VectSharp.PDF.PDFContextInterpreter.FilterOption Class Reference . . . 228
7.45.1 Detailed Description . . . 229
7.45.2 Member Enumeration Documentation . . . 229
7.45.2.1 FilterOperations . . . 229
7.45.3 Constructor & Destructor Documentation . . . 229
7.45.3.1 FilterOption() . . . 229
7.45.4 Member Data Documentation . . . 229
7.45.4.1 Default . . . 230
7.45.5 Property Documentation . . . 230
7.45.5.1 Operation . . . 230
7.45.5.2 RasterisationResolution . . . 230
7.45.5.3 RasterisationResolutionRelative . . . 230
7.46 VectSharp.SVG.SVGContextInterpreter.FilterOption Class Reference . . . 231
7.46.1 Detailed Description . . . 231
7.46.2 Member Enumeration Documentation . . . 231
7.46.2.1 FilterOperations . . . 231
7.46.3 Constructor & Destructor Documentation . . . 232
7.46.3.1 FilterOption() . . . 232
7.46.4 Member Data Documentation . . . 232
7.46.4.1 Default . . . 232
7.46.5 Property Documentation . . . 232
7.46.5.1 Operation . . . 233
7.46.5.2 RasterisationResolution . . . 233
7.46.5.3 RasterisationResolutionRelative . . . 233
7.47 VectSharp.Filters.FilterWithRasterisableParameter Class Reference . . . 234
7.47.1 Detailed Description . . . 234
7.47.2 Member Function Documentation . . . 234
7.47.2.1 Dispose() . . . 235
7.47.2.2 RasteriseParameter() . . . 235
7.48 VectSharp.FolderFontLibrary Class Reference . . . 235
7.48.1 Detailed Description . . . 236
7.48.2 Constructor & Destructor Documentation . . . 237
7.48.2.1 FolderFontLibrary() [1/2] . . . 237
7.48.2.2 FolderFontLibrary() [2/2] . . . 237

7.48.3 Member Function Documentation . . . 237
7.48.3.1 ResolveFontFamily() [1/2] . . . 237
7.48.3.2 ResolveFontFamily() [2/2] . . . 238
7.49 VectSharp.Font Class Reference . . . 238
7.49.1 Detailed Description . . . 239
7.49.2 Constructor & Destructor Documentation . . . 240
7.49.2.1 Font() [1/3] . . . 240
7.49.2.2 Font() [2/3] . . . 240
7.49.2.3 Font() [3/3] . . . 240
7.49.3 Member Function Documentation . . . 242
7.49.3.1 MeasureText() . . . 242
7.49.3.2 MeasureTextAdvanced() . . . 242
7.49.4 Member Data Documentation . . . 243
7.49.4.1 EnableKerning . . . 243
7.49.5 Property Documentation . . . 243
7.49.5.1 Ascent . . . 243
7.49.5.2 Descent . . . 243
7.49.5.3 FontFamily . . . 243
7.49.5.4 FontSize . . . 244
7.49.5.5 Underline . . . 244
7.49.5.6 WinAscent . . . 244
7.49.5.7 YMax . . . 244
7.49.5.8 YMin . . . 244
7.50 VectSharp.FontFamily Class Reference . . . 245
7.50.1 Detailed Description . . . 246
7.50.2 Member Enumeration Documentation . . . 246
7.50.2.1 StandardFontFamilies . . . 246
7.50.3 Constructor & Destructor Documentation . . . 247
7.50.3.1 FontFamily() [1/4] . . . 247
7.50.3.2 FontFamily() [2/4] . . . 247
7.50.3.3 FontFamily() [3/4] . . . 247
7.50.3.4 FontFamily() [4/4] . . . 248
7.50.4 Member Function Documentation . . . 248
7.50.4.1 ResolveFontFamily() [1/4] . . . 248
7.50.4.2 ResolveFontFamily() [2/4] . . . 248
7.50.4.3 ResolveFontFamily() [3/4] . . . 249
7.50.4.4 ResolveFontFamily() [4/4] . . . 249
7.50.5 Member Data Documentation . . . 250
7.50.5.1 StandardFamilies . . . 250
7.50.5.2 StandardFontFamilyResources . . . 250
7.50.6 Property Documentation . . . 250
7.50.6.1 DefaultFontLibrary . . . 250

7.50.6.2 FamilyName . . . 251
7.50.6.3 FileName . . . 251
7.50.6.4 IsBold . . . 251
7.50.6.5 IsItalic . . . 251
7.50.6.6 IsOblique . . . 251
7.50.6.7 IsStandardFamily . . . 252
7.50.6.8 TrueTypeFile . . . 252
7.51 VectSharp.FontFamilyCreationException Class Reference . . . 252
7.51.1 Detailed Description . . . 253
7.51.2 Constructor & Destructor Documentation . . . 253
7.51.2.1 FontFamilyCreationException() . . . 253
7.51.3 Property Documentation . . . 253
7.51.3.1 FontFamily . . . 253
7.52 VectSharp.FontLibrary Class Reference . . . 254
7.52.1 Detailed Description . . . 255
7.52.2 Member Function Documentation . . . 255
7.52.2.1 ResolveFontFamily() [1/4] . . . 255
7.52.2.2 ResolveFontFamily() [2/4] . . . 256
7.52.2.3 ResolveFontFamily() [3/4] . . . 256
7.52.2.4 ResolveFontFamily() [4/4] . . . 257
7.53 VectSharp.Font.FontUnderline Class Reference . . . 257
7.53.1 Detailed Description . . . 257
7.53.2 Property Documentation . . . 258
7.53.2.1 FollowItalicAngle . . . 258
7.53.2.2 LineCap . . . 258
7.53.2.3 Position . . . 258
7.53.2.4 SkipDescenders . . . 258
7.53.2.5 Thickness . . . 259
7.54 VectSharp.Markdown.FormattedString Struct Reference . . . 259
7.54.1 Detailed Description . . . 259
7.54.2 Constructor & Destructor Documentation . . . 259
7.54.2.1 FormattedString() . . . 259
7.54.3 Property Documentation . . . 260
7.54.3.1 Colour . . . 260
7.54.3.2 IsBold . . . 260
7.54.3.3 IsItalic . . . 260
7.54.3.4 Text . . . 261
7.55 VectSharp.FormattedText Class Reference . . . 261
7.55.1 Detailed Description . . . 262
7.55.2 Constructor & Destructor Documentation . . . 262
7.55.2.1 FormattedText() . . . 262
7.55.3 Member Function Documentation . . . 262

7.55.3.1 Format() [1/2] . . . 262
7.55.3.2 Format() [2/2] . . . 263
7.55.4 Property Documentation . . . 264
7.55.4.1 Brush . . . 264
7.55.4.2 Font . . . 265
7.55.4.3 Script . . . 265
7.55.4.4 Spacing . . . 265
7.55.4.5 Text . . . 265
7.56 VectSharp.FormattedTextExtensions Class Reference . . . 265
7.56.1 Detailed Description . . . 266
7.56.2 Member Function Documentation . . . 266
7.56.2.1 GetText() . . . 266
7.56.2.2 Measure() . . . 266
7.57 VectSharp.Frame Class Reference . . . 267
7.57.1 Detailed Description . . . 267
7.57.2 Constructor & Destructor Documentation . . . 267
7.57.2.1 Frame() . . . 267
7.57.3 Property Documentation . . . 268
7.57.3.1 Duration . . . 268
7.57.3.2 Graphics . . . 268
7.58 VectSharp.Plots.Function2D Class Reference . . . 268
7.58.1 Detailed Description . . . 269
7.58.2 Member Enumeration Documentation . . . 269
7.58.2.1 PlotType . . . 270
7.58.3 Constructor & Destructor Documentation . . . 270
7.58.3.1 Function2D() . . . 270
7.58.4 Member Function Documentation . . . 270
7.58.4.1 Plot() . . . 270
7.58.5 Property Documentation . . . 271
7.58.5.1 Colouring . . . 271
7.58.5.2 CoordinateSystem . . . 271
7.58.5.3 Function . . . 271
7.58.5.4 RasterResolutionX . . . 271
7.58.5.5 RasterResolutionY . . . 272
7.58.5.6 SampledPointElement . . . 272
7.58.5.7 Tag . . . 272
7.58.5.8 Type . . . 272
7.59 VectSharp.Plots.Function2DGrid Class Reference . . . 272
7.59.1 Detailed Description . . . 273
7.59.2 Member Enumeration Documentation . . . 273
7.59.2.1 GridType . . . 274
7.59.3 Constructor & Destructor Documentation . . . 274

7.59.3.1 Function2DGrid() [1/2] . . . 274
7.59.3.2 Function2DGrid() [2/2] . . . 274
7.59.4 Member Function Documentation . . . 275
7.59.4.1 ToRectangular() . . . 275
7.59.5 Property Documentation . . . 275
7.59.5.1 DataPoints . . . 275
7.59.5.2 MaxX . . . 276
7.59.5.3 MaxY . . . 276
7.59.5.4 MaxZ . . . 276
7.59.5.5 MinX . . . 276
7.59.5.6 MinY . . . 276
7.59.5.7 MinZ . . . 277
7.59.5.8 StepsX . . . 277
7.59.5.9 StepsY . . . 277
7.59.5.10 Type . . . 277
7.60 VectSharp.Filters.GaussianBlurFilter Class Reference . . . 278
7.60.1 Detailed Description . . . 279
7.60.2 Constructor & Destructor Documentation . . . 279
7.60.2.1 GaussianBlurFilter() . . . 279
7.60.3 Member Function Documentation . . . 279
7.60.3.1 Filter() . . . 279
7.60.4 Property Documentation . . . 280
7.60.4.1 BottomRightMargin . . . 280
7.60.4.2 StandardDeviation . . . 280
7.60.4.3 TopLeftMargin . . . 280
7.61 VectSharp.GradientBrush Class Reference . . . 281
7.61.1 Detailed Description . . . 281
7.61.2 Property Documentation . . . 281
7.61.2.1 GradientStops . . . 281
7.62 VectSharp.Gradients Class Reference . . . 282
7.62.1 Detailed Description . . . 284
7.62.2 Member Function Documentation . . . 284
7.62.2.1 CividisColouring() . . . 284
7.62.2.2 InfernoColouring() . . . 284
7.62.2.3 MagmaColouring() . . . 285
7.62.2.4 MakoColouring() . . . 285
7.62.2.5 PlasmaColouring() . . . 285
7.62.2.6 RocketColouring() . . . 286
7.62.2.7 TurboColouring() . . . 286
7.62.2.8 ViridisColouring() . . . 287
7.62.3 Member Data Documentation . . . 287
7.62.3.1 Cividis . . . 287

7.62.3.2 Inferno . . . 287
7.62.3.3 Magma . . . 288
7.62.3.4 Mako . . . 288
7.62.3.5 MutedRainbow . . . 288
7.62.3.6 MutedRainbowDiscrete . . . 288
7.62.3.7 OkabeItoRainbow . . . 289
7.62.3.8 OkabeItoRainbowDiscrete . . . 289
7.62.3.9 Plasma . . . 289
7.62.3.10 Rainbow . . . 289
7.62.3.11 RedToGreen . . . 290
7.62.3.12 RedYellowGreen . . . 290
7.62.3.13 Rocket . . . 290
7.62.3.14 TransparentToBlack . . . 290
7.62.3.15 Turbo . . . 291
7.62.3.16 Viridis . . . 291
7.62.3.17 WhiteToBlack . . . 291
7.63 VectSharp.GradientStop Struct Reference . . . 291
7.63.1 Detailed Description . . . 292
7.63.2 Constructor & Destructor Documentation . . . 292
7.63.2.1 GradientStop() . . . 292
7.63.3 Member Function Documentation . . . 292
7.63.3.1 MultiplyOpacity() . . . 292
7.63.4 Property Documentation . . . 293
7.63.4.1 Colour . . . 293
7.63.4.2 Offset . . . 293
7.64 VectSharp.GradientStops Class Reference . . . 293
7.64.1 Detailed Description . . . 294
7.64.2 Constructor & Destructor Documentation . . . 294
7.64.2.1 GradientStops() [1/2] . . . 294
7.64.2.2 GradientStops() [2/2] . . . 295
7.64.3 Member Function Documentation . . . 295
7.64.3.1 GetColourAt() . . . 295
7.64.3.2 GetEnumerator() . . . 295
7.64.3.3 operator Func< double, Colour >() . . . 296
7.64.4 Member Data Documentation . . . 296
7.64.4.1 StopTolerance . . . 296
7.64.5 Property Documentation . . . 296
7.64.5.1 Count . . . 296
7.64.5.2 this[int index] . . . 296
7.65 VectSharp.Graphics Class Reference . . . 297
7.65.1 Detailed Description . . . 301
7.65.2 Member Function Documentation . . . 301

7.65.2.1 CopyToIGraphicsContext() . . . 301
7.65.2.2 Crop() [1/2] . . . 301
7.65.2.3 Crop() [2/2] . . . 301
7.65.2.4 DrawGraphics() [1/6] . . . 302
7.65.2.5 DrawGraphics() [2/6] . . . 302
7.65.2.6 DrawGraphics() [3/6] . . . 303
7.65.2.7 DrawGraphics() [4/6] . . . 303
7.65.2.8 DrawGraphics() [5/6] . . . 303
7.65.2.9 DrawGraphics() [6/6] . . . 304
7.65.2.10 DrawRasterImage() [1/5] . . . 304
7.65.2.11 DrawRasterImage() [2/5] . . . 305
7.65.2.12 DrawRasterImage() [3/5] . . . 305
7.65.2.13 DrawRasterImage() [4/5] . . . 306
7.65.2.14 DrawRasterImage() [5/5] . . . 306
7.65.2.15 FillPath() [1/2] . . . 307
7.65.2.16 FillPath() [2/2] . . . 307
7.65.2.17 FillRectangle() [1/2] . . . 307
7.65.2.18 FillRectangle() [2/2] . . . 308
7.65.2.19 FillText() [1/6] . . . 308
7.65.2.20 FillText() [2/6] . . . 310
7.65.2.21 FillText() [3/6] . . . 310
7.65.2.22 FillText() [4/6] . . . 312
7.65.2.23 FillText() [5/6] . . . 312
7.65.2.24 FillText() [6/6] . . . 313
7.65.2.25 FillTextOnPath() . . . 313
7.65.2.26 FillTextUnderline() [1/4] . . . 315
7.65.2.27 FillTextUnderline() [2/4] . . . 315
7.65.2.28 FillTextUnderline() [3/4] . . . 317
7.65.2.29 FillTextUnderline() [4/4] . . . 317
7.65.2.30 GetBounds() . . . 318
7.65.2.31 GetTags() . . . 318
7.65.2.32 Linearise() . . . 318
7.65.2.33 MeasureText() [1/2] . . . 319
7.65.2.34 MeasureText() [2/2] . . . 319
7.65.2.35 Restore() . . . 320
7.65.2.36 Rotate() . . . 320
7.65.2.37 RotateAt() . . . 320
7.65.2.38 Save() . . . 321
7.65.2.39 Scale() . . . 321
7.65.2.40 SetClippingPath() [1/3] . . . 321
7.65.2.41 SetClippingPath() [2/3] . . . 322
7.65.2.42 SetClippingPath() [3/3] . . . 322

7.65.2.43 StrokePath() . . . 323
7.65.2.44 StrokeRectangle() [1/2] . . . 323
7.65.2.45 StrokeRectangle() [2/2] . . . 324
7.65.2.46 StrokeText() [1/6] . . . 324
7.65.2.47 StrokeText() [2/6] . . . 325
7.65.2.48 StrokeText() [3/6] . . . 326
7.65.2.49 StrokeText() [4/6] . . . 326
7.65.2.50 StrokeText() [5/6] . . . 327
7.65.2.51 StrokeText() [6/6] . . . 328
7.65.2.52 StrokeTextOnPath() . . . 328
7.65.2.53 StrokeTextUnderline() [1/4] . . . 329
7.65.2.54 StrokeTextUnderline() [2/4] . . . 330
7.65.2.55 StrokeTextUnderline() [3/4] . . . 330
7.65.2.56 StrokeTextUnderline() [4/4] . . . 331
7.65.2.57 Transform() [1/3] . . . 332
7.65.2.58 Transform() [2/3] . . . 332
7.65.2.59 Transform() [3/3] . . . 333
7.65.2.60 Translate() [1/2] . . . 333
7.65.2.61 Translate() [2/2] . . . 333
7.65.2.62 TryRasterise() . . . 334
7.65.3 Member Data Documentation . . . 334
7.65.3.1 RasterisationMethod . . . 334
7.65.4 Property Documentation . . . 335
7.65.4.1 DefaultFillRule . . . 335
7.65.4.2 UnbalancedStackAction . . . 335
7.65.4.3 UseUniqueTags . . . 335
7.66 VectSharp.Plots.GraphicsDataPointElement Class Reference . . . 335
7.66.1 Detailed Description . . . 336
7.66.2 Constructor & Destructor Documentation . . . 336
7.66.2.1 GraphicsDataPointElement() . . . 336
7.66.3 Member Function Documentation . . . 336
7.66.3.1 Plot() . . . 337
7.66.4 Property Documentation . . . 337
7.66.4.1 Graphics . . . 337
7.67 VectSharp.GraphicsPath Class Reference . . . 337
7.67.1 Detailed Description . . . 339
7.67.2 Member Function Documentation . . . 340
7.67.2.1 AddPath() . . . 340
7.67.2.2 AddSmoothSpline() . . . 340
7.67.2.3 AddText() [1/2] . . . 340
7.67.2.4 AddText() [2/2] . . . 342
7.67.2.5 AddTextOnPath() . . . 342

7.67.2.6 AddTextUnderline() . . . 343
7.67.2.7 Arc() [1/2] . . . 344
7.67.2.8 Arc() [2/2] . . . 344
7.67.2.9 Close() . . . 345
7.67.2.10 ContainsPoint() . . . 345
7.67.2.11 CubicBezierTo() [1/2] . . . 345
7.67.2.12 CubicBezierTo() [2/2] . . . 346
7.67.2.13 Discretise() . . . 346
7.67.2.14 EllipticalArc() . . . 347
7.67.2.15 Flatten() . . . 347
7.67.2.16 GetBounds() . . . 348
7.67.2.17 GetFigures() . . . 348
7.67.2.18 GetLinearisationPointsNormals() . . . 348
7.67.2.19 GetNormalAtAbsolute() . . . 349
7.67.2.20 GetNormalAtRelative() . . . 349
7.67.2.21 GetPointAtAbsolute() . . . 350
7.67.2.22 GetPointAtRelative() . . . 350
7.67.2.23 GetPoints() . . . 350
7.67.2.24 GetStroke() . . . 351
7.67.2.25 GetTangentAtAbsolute() . . . 351
7.67.2.26 GetTangentAtRelative() . . . 351
7.67.2.27 Linearise() . . . 352
7.67.2.28 LineTo() [1/2] . . . 352
7.67.2.29 LineTo() [2/2] . . . 353
7.67.2.30 MeasureLength() . . . 353
7.67.2.31 MoveTo() [1/2] . . . 353
7.67.2.32 MoveTo() [2/2] . . . 354
7.67.2.33 QuadraticBezierTo() [1/2] . . . 354
7.67.2.34 QuadraticBezierTo() [2/2] . . . 355
7.67.2.35 Reverse() . . . 355
7.67.2.36 Transform() . . . 355
7.67.2.37 Triangulate() . . . 356
7.67.3 Property Documentation . . . 356
7.67.3.1 Segments . . . 356
7.68 VectSharp.Plots.Grid Class Reference . . . 357
7.68.1 Detailed Description . . . 358
7.68.2 Constructor & Destructor Documentation . . . 358
7.68.2.1 Grid() . . . 358
7.68.3 Member Function Documentation . . . 358
7.68.3.1 Plot() . . . 358
7.68.4 Property Documentation . . . 359
7.68.4.1 CoordinateSystem . . . 359

7.68.4.2 IntervalCount . . . 359
7.68.4.3 PresentationAttributes . . . 359
7.68.4.4 Side1End . . . 359
7.68.4.5 Side1Start . . . 360
7.68.4.6 Side2End . . . 360
7.68.4.7 Side2Start . . . 360
7.68.4.8 Tag . . . 360
7.69 VectSharp.Markdown.HTTPUtils Class Reference . . . 360
7.69.1 Detailed Description . . . 361
7.69.2 Member Function Documentation . . . 361
7.69.2.1 ResolveEmojiUsingOpenMoji() . . . 361
7.69.2.2 ResolveImageURI() . . . 361
7.69.3 Member Data Documentation . . . 361
7.69.3.1 path . . . 361
7.69.4 Property Documentation . . . 362
7.69.4.1 LogDownloads . . . 362
7.70 VectSharp.Plots.IContinuousCoordinateSystem Interface Reference . . . 362
7.70.1 Detailed Description . . . 363
7.70.2 Member Function Documentation . . . 363
7.70.2.1 GetAround() . . . 363
7.70.2.2 IsDirectionStraight() . . . 364
7.70.3 Property Documentation . . . 364
7.70.3.1 IsLinear . . . 364
7.70.3.2 Resolution . . . 364
7.71 VectSharp.Plots.IContinuousInvertibleCoordinateSystem Interface Reference . . . 365
7.71.1 Detailed Description . . . 365
7.71.2 Member Function Documentation . . . 365
7.71.2.1 ToDataCoordinates() . . . 365
7.72 VectSharp.Plots.ICoordinateSystem< T > Interface Template Reference . . . 366
7.72.1 Detailed Description . . . 366
7.72.2 Member Function Documentation . . . 367
7.72.2.1 ToPlotCoordinates() . . . 367
7.73 VectSharp.Plots.ICoordinateSystem1D< T > Interface Template Reference . . . 367
7.73.1 Detailed Description . . . 368
7.73.2 Member Function Documentation . . . 368
7.73.2.1 ToPlotCoordinates() . . . 368
7.74 VectSharp.Plots.IDataPointElement Interface Reference . . . 368
7.74.1 Detailed Description . . . 369
7.74.2 Member Function Documentation . . . 369
7.74.2.1 Plot() . . . 369
7.75 VectSharp.IEasing Interface Reference . . . 370
7.75.1 Detailed Description . . . 370

7.75.2 Member Function Documentation . . . 370
7.75.2.1 Ease() . . . 370
7.76 VectSharp.Filters.IFilter Interface Reference . . . 371
7.76.1 Detailed Description . . . 372
7.76.2 Property Documentation . . . 372
7.76.2.1 BottomRightMargin . . . 372
7.76.2.2 TopLeftMargin . . . 372
7.77 VectSharp.Filters.IFilterWithLocation Interface Reference . . . 373
7.77.1 Detailed Description . . . 373
7.77.2 Member Function Documentation . . . 373
7.77.2.1 Filter() . . . 373
7.78 VectSharp.Filters.IFilterWithRasterisableParameter Interface Reference . . . 374
7.78.1 Detailed Description . . . 375
7.78.2 Member Function Documentation . . . 375
7.78.2.1 RasteriseParameter() . . . 375
7.79 VectSharp.IFontLibrary Interface Reference . . . 375
7.79.1 Detailed Description . . . 376
7.79.2 Member Function Documentation . . . 376
7.79.2.1 ResolveFontFamily() [1/4] . . . 376
7.79.2.2 ResolveFontFamily() [2/4] . . . 377
7.79.2.3 ResolveFontFamily() [3/4] . . . 377
7.79.2.4 ResolveFontFamily() [4/4] . . . 377
7.80 VectSharp.IGraphicsContext Interface Reference . . . 378
7.80.1 Detailed Description . . . 380
7.80.2 Member Function Documentation . . . 380
7.80.2.1 CubicBezierTo() . . . 380
7.80.2.2 DrawFilteredGraphics() . . . 380
7.80.2.3 DrawRasterImage() . . . 381
7.80.2.4 Fill() . . . 381
7.80.2.5 FillText() . . . 381
7.80.2.6 LineTo() . . . 382
7.80.2.7 MoveTo() . . . 382
7.80.2.8 Rectangle() . . . 382
7.80.2.9 Rotate() . . . 383
7.80.2.10 Scale() . . . 383
7.80.2.11 SetFillStyle() [1/2] . . . 383
7.80.2.12 SetFillStyle() [2/2] . . . 384
7.80.2.13 SetLineDash() . . . 384
7.80.2.14 SetStrokeStyle() [1/2] . . . 384
7.80.2.15 SetStrokeStyle() [2/2] . . . 385
7.80.2.16 StrokeText() . . . 385
7.80.2.17 Transform() . . . 385

7.80.2.18 Translate() . . . 386
7.80.3 Property Documentation . . . 386
7.80.3.1 FillStyle . . . 386
7.80.3.2 Font . . . 386
7.80.3.3 Height . . . 386
7.80.3.4 LineCap . . . 387
7.80.3.5 LineJoin . . . 387
7.80.3.6 LineWidth . . . 387
7.80.3.7 StrokeStyle . . . 387
7.80.3.8 Tag . . . 387
7.80.3.9 TextBaseline . . . 388
7.80.3.10 Width . . . 388
7.81 VectSharp.ThreeD.ILightSource Interface Reference . . . 388
7.81.1 Detailed Description . . . 389
7.81.2 Member Function Documentation . . . 389
7.81.2.1 GetLightAt() . . . 389
7.81.2.2 GetObstruction() . . . 389
7.81.3 Property Documentation . . . 390
7.81.3.1 CastsShadow . . . 390
7.82 VectSharp.Filters.ILocationInvariantFilter Interface Reference . . . 390
7.82.1 Detailed Description . . . 391
7.82.2 Member Function Documentation . . . 391
7.82.2.1 Filter() . . . 391
7.83 VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter Class Reference . . . 391
7.83.1 Detailed Description . . . 392
7.83.2 Member Function Documentation . . . 392
7.83.2.1 Rasterise() . . . 393
7.83.2.2 SaveAsAnimatedGIF() [1/3] . . . 393
7.83.2.3 SaveAsAnimatedGIF() [2/3] . . . 394
7.83.2.4 SaveAsAnimatedGIF() [3/3] . . . 394
7.83.2.5 SaveAsAnimatedPNG() [1/2] . . . 395
7.83.2.6 SaveAsAnimatedPNG() [2/2] . . . 395
7.83.2.7 SaveAsImage() [1/3] . . . 396
7.83.2.8 SaveAsImage() [2/3] . . . 396
7.83.2.9 SaveAsImage() [3/3] . . . 397
7.83.2.10 SaveAsRawBytes() [1/2] . . . 397
7.83.2.11 SaveAsRawBytes() [2/2] . . . 398
7.84 VectSharp.ImageSharpUtils.ImageURIParser Class Reference . . . 398
7.84.1 Detailed Description . . . 399
7.84.2 Member Function Documentation . . . 399
7.84.2.1 Parser() . . . 399
7.85 VectSharp.MuPDFUtils.ImageURIParser Class Reference . . . 399

7.85.1 Detailed Description . . . 400
7.85.2 Member Function Documentation . . . 400
7.85.2.1 Parser() . . . 400
7.86 VectSharp.ThreeD.IMaterial Interface Reference . . . 400
7.86.1 Detailed Description . . . 401
7.86.2 Member Function Documentation . . . 401
7.86.2.1 GetColour() . . . 401
7.87 VectSharp.Markdown.InlineRenderedEventArgs Class Reference . . . 402
7.87.1 Detailed Description . . . 402
7.87.2 Constructor & Destructor Documentation . . . 403
7.87.2.1 InlineRenderedEventArgs() . . . 403
7.87.3 Property Documentation . . . 403
7.87.3.1 Graphics . . . 403
7.87.3.2 Inline . . . 403
7.87.3.3 MarkdownContext . . . 404
7.88 VectSharp.Markdown.InlineRenderingEventArgs Class Reference . . . 404
7.88.1 Detailed Description . . . 405
7.88.2 Constructor & Destructor Documentation . . . 405
7.88.2.1 InlineRenderingEventArgs() . . . 405
7.88.3 Property Documentation . . . 405
7.88.3.1 Graphics . . . 405
7.88.3.2 Inline . . . 405
7.88.3.3 MarkdownContext . . . 406
7.89 VectSharp.Plots.IPlotElement Interface Reference . . . 406
7.89.1 Detailed Description . . . 408
7.89.2 Member Function Documentation . . . 408
7.89.2.1 Plot() . . . 408
7.89.3 Property Documentation . . . 408
7.89.3.1 CoordinateSystem . . . 408
7.90 VectSharp.ThreeD.IScene Interface Reference . . . 409
7.90.1 Detailed Description . . . 410
7.90.2 Member Function Documentation . . . 410
7.90.2.1 AddElement() . . . 410
7.90.2.2 AddRange() . . . 410
7.90.2.3 Replace() [1/2] . . . 410
7.90.2.4 Replace() [2/2] . . . 411
7.90.3 Property Documentation . . . 411
7.90.3.1 SceneElements . . . 411
7.90.3.2 SceneLock . . . 411
7.91 VectSharp.ThreeD.LightIntensity Struct Reference . . . 412
7.91.1 Detailed Description . . . 412
7.91.2 Constructor & Destructor Documentation . . . 412

7.91.2.1 LightIntensity() . . . 412
7.91.3 Member Function Documentation . . . 413
7.91.3.1 Deconstruct() . . . 413
7.91.4 Member Data Documentation . . . 413
7.91.4.1 Direction . . . 413
7.91.4.2 Intensity . . . 413
7.92 VectSharp.Markdown.Line Class Reference . . . 414
7.92.1 Detailed Description . . . 414
7.92.2 Constructor & Destructor Documentation . . . 414
7.92.2.1 Line() . . . 414
7.92.3 Property Documentation . . . 414
7.92.3.1 Fragments . . . 415
7.92.3.2 InitialAscent . . . 415
7.93 VectSharp.Plots.LinearCoordinateSystem1D Class Reference . . . 415
7.93.1 Detailed Description . . . 416
7.93.2 Constructor & Destructor Documentation . . . 416
7.93.2.1 LinearCoordinateSystem1D() [1/2] . . . 416
7.93.2.2 LinearCoordinateSystem1D() [2/2] . . . 416
7.93.3 Member Function Documentation . . . 417
7.93.3.1 ToPlotCoordinates() . . . 417
7.93.4 Property Documentation . . . 417
7.93.4.1 Max . . . 417
7.93.4.2 Min . . . 417
7.93.4.3 Scale . . . 418
7.94 VectSharp.Plots.LinearCoordinateSystem2D Class Reference . . . 418
7.94.1 Detailed Description . . . 420
7.94.2 Constructor & Destructor Documentation . . . 420
7.94.2.1 LinearCoordinateSystem2D() [1/3] . . . 420
7.94.2.2 LinearCoordinateSystem2D() [2/3] . . . 420
7.94.2.3 LinearCoordinateSystem2D() [3/3] . . . 421
7.94.3 Member Function Documentation . . . 421
7.94.3.1 GetAround() . . . 421
7.94.3.2 IsDirectionStraight() . . . 422
7.94.3.3 ToDataCoordinates() . . . 422
7.94.3.4 ToPlotCoordinates() [1/2] . . . 422
7.94.3.5 ToPlotCoordinates() [2/2] . . . 423
7.94.4 Property Documentation . . . 424
7.94.4.1 IsLinear . . . 424
7.94.4.2 MaxX . . . 424
7.94.4.3 MaxY . . . 424
7.94.4.4 MinX . . . 425
7.94.4.5 MinY . . . 425

7.94.4.6 Resolution . . . 425
7.94.4.7 ScaleX . . . 425
7.94.4.8 ScaleY . . . 425
7.95 VectSharp.LinearEasing Class Reference . . . 426
7.95.1 Detailed Description . . . 426
7.95.2 Constructor & Destructor Documentation . . . 426
7.95.2.1 LinearEasing() . . . 426
7.95.3 Member Function Documentation . . . 427
7.95.3.1 Ease() . . . 427
7.96 VectSharp.LinearGradientBrush Class Reference . . . 427
7.96.1 Detailed Description . . . 428
7.96.2 Constructor & Destructor Documentation . . . 428
7.96.2.1 LinearGradientBrush() [1/2] . . . 428
7.96.2.2 LinearGradientBrush() [2/2] . . . 429
7.96.3 Member Function Documentation . . . 429
7.96.3.1 MultiplyOpacity() . . . 429
7.96.3.2 RelativeTo() . . . 430
7.96.4 Property Documentation . . . 430
7.96.4.1 EndPoint . . . 430
7.96.4.2 StartPoint . . . 431
7.97 VectSharp.Plots.LinearTrendLine Class Reference . . . 431
7.97.1 Detailed Description . . . 432
7.97.2 Constructor & Destructor Documentation . . . 432
7.97.2.1 LinearTrendLine() [1/3] . . . 432
7.97.2.2 LinearTrendLine() [2/3] . . . 433
7.97.2.3 LinearTrendLine() [3/3] . . . 433
7.97.3 Member Function Documentation . . . 433
7.97.3.1 Plot() . . . 434
7.97.4 Property Documentation . . . 434
7.97.4.1 CoordinateSystem . . . 434
7.97.4.2 Intercept . . . 434
7.97.4.3 MaxX . . . 434
7.97.4.4 MaxY . . . 435
7.97.4.5 MinX . . . 435
7.97.4.6 MinY . . . 435
7.97.4.7 PresentationAttributes . . . 435
7.97.4.8 Slope . . . 435
7.97.4.9 Tag . . . 436
7.98 VectSharp.LineDash Struct Reference . . . 436
7.98.1 Detailed Description . . . 436
7.98.2 Constructor & Destructor Documentation . . . 437
7.98.2.1 LineDash() [1/3] . . . 437

7.98.2.2 LineDash() [2/3] . . . 437
7.98.2.3 LineDash() [3/3] . . . 437
7.98.3 Member Data Documentation . . . 438
7.98.3.1 DashArray . . . 438
7.98.3.2 Phase . . . 438
7.98.3.3 SolidLine . . . 438
7.98.3.4 UnitsOff . . . 438
7.98.3.5 UnitsOn . . . 439
7.99 VectSharp.Markdown.LineEventArgs Class Reference . . . 439
7.99.1 Detailed Description . . . 440
7.99.2 Constructor & Destructor Documentation . . . 440
7.99.2.1 LineEventArgs() . . . 440
7.99.3 Property Documentation . . . 441
7.99.3.1 Baseline . . . 441
7.99.3.2 ContentMaxX . . . 441
7.99.3.3 ContentMaxY . . . 441
7.99.3.4 ContentMinY . . . 441
7.99.3.5 Graphics . . . 441
7.99.3.6 Line . . . 442
7.99.3.7 MarkdownContext . . . 442
7.99.3.8 PageMaxX . . . 442
7.99.3.9 PageMaxY . . . 442
7.100 VectSharp.Markdown.LineFragment Class Reference . . . 443
7.100.1 Detailed Description . . . 443
7.100.2 Member Function Documentation . . . 443
7.100.2.1 GetAscent() . . . 443
7.100.2.2 GetMaxX() . . . 444
7.100.2.3 GetMaxY() . . . 444
7.100.2.4 Render() . . . 444
7.100.2.5 Translate() . . . 445
7.100.3 Property Documentation . . . 445
7.100.3.1 Tag . . . 445
7.101 VectSharp.Markdown.LineMeasuredEventArgs Class Reference . . . 446
7.101.1 Detailed Description . . . 447
7.101.2 Constructor & Destructor Documentation . . . 447
7.101.2.1 LineMeasuredEventArgs() . . . 447
7.101.3 Property Documentation . . . 447
7.101.3.1 Baseline . . . 447
7.101.3.2 ContentMaxX . . . 448
7.101.3.3 ContentMaxY . . . 448
7.101.3.4 ContentMinY . . . 448
7.101.3.5 Graphics . . . 448

7.101.3.6 Line . . . 448
7.101.3.7 MarkdownContext . . . 449
7.101.3.8 PageMaxX . . . 449
7.101.3.9 PageMaxY . . . 449
7.102 VectSharp.Plots.LinLogCoordinateSystem2D Class Reference . . . 449
7.102.1 Detailed Description . . . 451
7.102.2 Constructor & Destructor Documentation . . . 451
7.102.2.1 LinLogCoordinateSystem2D() [1/3] . . . 452
7.102.2.2 LinLogCoordinateSystem2D() [2/3] . . . 452
7.102.2.3 LinLogCoordinateSystem2D() [3/3] . . . 452
7.102.3 Member Function Documentation . . . 453
7.102.3.1 GetAround() . . . 453
7.102.3.2 IsDirectionStraight() . . . 453
7.102.3.3 ToDataCoordinates() . . . 454
7.102.3.4 ToPlotCoordinates() [1/2] . . . 454
7.102.3.5 ToPlotCoordinates() [2/2] . . . 454
7.102.4 Property Documentation . . . 455
7.102.4.1 IsLinear . . . 455
7.102.4.2 MaxX . . . 455
7.102.4.3 MaxY . . . 455
7.102.4.4 MinX . . . 456
7.102.4.5 MinY . . . 456
7.102.4.6 Resolution . . . 456
7.102.4.7 ScaleX . . . 456
7.102.4.8 ScaleY . . . 456
7.103 VectSharp.Plots.LogarithmicCoordinateSystem1D Class Reference . . . 457
7.103.1 Detailed Description . . . 457
7.103.2 Constructor & Destructor Documentation . . . 457
7.103.2.1 LogarithmicCoordinateSystem1D() [1/2] . . . 458
7.103.2.2 LogarithmicCoordinateSystem1D() [2/2] . . . 458
7.103.3 Member Function Documentation . . . 458
7.103.3.1 ToPlotCoordinates() . . . 458
7.103.4 Property Documentation . . . 458
7.103.4.1 Max . . . 459
7.103.4.2 Min . . . 459
7.103.4.3 Scale . . . 459
7.104 VectSharp.Plots.LogarithmicCoordinateSystem2D Class Reference . . . 459
7.104.1 Detailed Description . . . 461
7.104.2 Constructor & Destructor Documentation . . . 461
7.104.2.1 LogarithmicCoordinateSystem2D() [1/3] . . . 462
7.104.2.2 LogarithmicCoordinateSystem2D() [2/3] . . . 462
7.104.2.3 LogarithmicCoordinateSystem2D() [3/3] . . . 462

7.104.3 Member Function Documentation . . . 463
7.104.3.1 GetAround() . . . 463
7.104.3.2 IsDirectionStraight() . . . 463
7.104.3.3 ToDataCoordinates() . . . 464
7.104.3.4 ToPlotCoordinates() [1/2] . . . 464
7.104.3.5 ToPlotCoordinates() [2/2] . . . 464
7.104.4 Property Documentation . . . 465
7.104.4.1 IsLinear . . . 465
7.104.4.2 MaxX . . . 465
7.104.4.3 MaxY . . . 465
7.104.4.4 MinX . . . 466
7.104.4.5 MinY . . . 466
7.104.4.6 Resolution . . . 466
7.104.4.7 ScaleX . . . 466
7.104.4.8 ScaleY . . . 466
7.105 VectSharp.Plots.LogarithmicTrendLine Class Reference . . . 467
7.105.1 Detailed Description . . . 468
7.105.2 Constructor & Destructor Documentation . . . 468
7.105.2.1 LogarithmicTrendLine() [1/3] . . . 468
7.105.2.2 LogarithmicTrendLine() [2/3] . . . 469
7.105.2.3 LogarithmicTrendLine() [3/3] . . . 469
7.105.3 Member Function Documentation . . . 469
7.105.3.1 Plot() . . . 469
7.105.4 Property Documentation . . . 470
7.105.4.1 CoordinateSystem . . . 470
7.105.4.2 Intercept . . . 470
7.105.4.3 MaxX . . . 470
7.105.4.4 MaxY . . . 471
7.105.4.5 MinX . . . 471
7.105.4.6 MinY . . . 471
7.105.4.7 PresentationAttributes . . . 471
7.105.4.8 Slope . . . 471
7.105.4.9 Tag . . . 472
7.106 VectSharp.Plots.LogLinCoordinateSystem2D Class Reference . . . 472
7.106.1 Detailed Description . . . 474
7.106.2 Constructor & Destructor Documentation . . . 474
7.106.2.1 LogLinCoordinateSystem2D() [1/3] . . . 474
7.106.2.2 LogLinCoordinateSystem2D() [2/3] . . . 474
7.106.2.3 LogLinCoordinateSystem2D() [3/3] . . . 475
7.106.3 Member Function Documentation . . . 475
7.106.3.1 GetAround() . . . 475
7.106.3.2 IsDirectionStraight() . . . 476

7.106.3.3 ToDataCoordinates() . . . 476
7.106.3.4 ToPlotCoordinates() [1/2] . . . 476
7.106.3.5 ToPlotCoordinates() [2/2] . . . 477
7.106.4 Property Documentation . . . 477
7.106.4.1 IsLinear . . . 477
7.106.4.2 MaxX . . . 477
7.106.4.3 MaxY . . . 477
7.106.4.4 MinX . . . 478
7.106.4.5 MinY . . . 478
7.106.4.6 Resolution . . . 478
7.106.4.7 ScaleX . . . 478
7.106.4.8 ScaleY . . . 478
7.107 VectSharp.Markdown.Margins Class Reference . . . 479
7.107.1 Detailed Description . . . 479
7.107.2 Constructor & Destructor Documentation . . . 479
7.107.2.1 Margins() . . . 479
7.107.3 Property Documentation . . . 480
7.107.3.1 Bottom . . . 480
7.107.3.2 Left . . . 480
7.107.3.3 Right . . . 480
7.107.3.4 Top . . . 480
7.108 VectSharp.MarkdownCanvas.MarkdownCanvasControl Class Reference . . . 481
7.108.1 Detailed Description . . . 482
7.108.2 Constructor & Destructor Documentation . . . 482
7.108.2.1 MarkdownCanvasControl() . . . 482
7.108.3 Member Data Documentation . . . 482
7.108.3.1 DocumentProperty . . . 483
7.108.3.2 DocumentSourceProperty . . . 483
7.108.3.3 MaxRenderWidthProperty . . . 483
7.108.3.4 MinRenderWidthProperty . . . 483
7.108.3.5 MinVariationProperty . . . 484
7.108.3.6 TextConversionOptionsProperty . . . 484
7.108.4 Property Documentation . . . 484
7.108.4.1 Document . . . 484
7.108.4.2 DocumentSource . . . 484
7.108.4.3 MaxRenderWidth . . . 485
7.108.4.4 MinRenderWidth . . . 485
7.108.4.5 MinVariation . . . 485
7.108.4.6 Renderer . . . 485
7.108.4.7 TextConversionOption . . . 486
7.109 VectSharp.Markdown.MarkdownContext Class Reference . . . 486
7.109.1 Detailed Description . . . 487

7.109.2 Member Function Documentation . . . 487
7.109.2.1 Clone() . . . 487
7.109.2.2 GetMaxX() [1/2] . . . 488
7.109.2.3 GetMaxX() [2/2] . . . 488
7.109.2.4 GetMinX() [1/2] . . . 488
7.109.2.5 GetMinX() [2/2] . . . 489
7.109.2.6 StartNewPage() . . . 489
7.109.3 Property Documentation . . . 490
7.109.3.1 BottomRight . . . 490
7.109.3.2 Colour . . . 490
7.109.3.3 CurrentLine . . . 490
7.109.3.4 CurrentPage . . . 490
7.109.3.5 Cursor . . . 490
7.109.3.6 Font . . . 491
7.109.3.7 ForbiddenAreasLeft . . . 491
7.109.3.8 ForbiddenAreasRight . . . 491
7.109.3.9 Headings . . . 491
7.109.3.10 InternalAnchors . . . 492
7.109.3.11 LinkDestinations . . . 492
7.109.3.12 ListDepth . . . 492
7.109.3.13 MarginBottomRight . . . 492
7.109.3.14 StrikeThrough . . . 492
7.109.3.15 Tag . . . 493
7.109.3.16 Translation . . . 493
7.109.3.17 Underline . . . 493
7.110 VectSharp.Markdown.MarkdownEmojiExtension Class Reference . . . 493
7.110.1 Detailed Description . . . 493
7.110.2 Member Function Documentation . . . 493
7.110.2.1 UseUnicodeEmoji() . . . 493
7.111 VectSharp.Markdown.MarkdownRenderer Class Reference . . . 494
7.111.1 Detailed Description . . . 498
7.111.2 Member Enumeration Documentation . . . 498
7.111.2.1 VerticalAlignment . . . 498
7.111.3 Constructor & Destructor Documentation . . . 499
7.111.3.1 MarkdownRenderer() . . . 499
7.111.4 Member Function Documentation . . . 499
7.111.4.1 Render() [1/2] . . . 499
7.111.4.2 Render() [2/2] . . . 499
7.111.4.3 RenderSinglePage() [1/2] . . . 500
7.111.4.4 RenderSinglePage() [2/2] . . . 501
7.111.5 Property Documentation . . . 501
7.111.5.1 AllowPageBreak . . . 501

7.111.5.2 BackgroundColour . . . 501
7.111.5.3 BaseFontSize . . . 502
7.111.5.4 BaseImageUri . . . 502
7.111.5.5 BaseLinkUri . . . 502
7.111.5.6 BoldFontFamily . . . 502
7.111.5.7 BoldItalicFontFamily . . . 502
7.111.5.8 BoldUnderlineThickness . . . 503
7.111.5.9 Bullets . . . 503
7.111.5.10 CodeBlockBackgroundColour . . . 503
7.111.5.11 CodeFont . . . 503
7.111.5.12 CodeFontBold . . . 504
7.111.5.13 CodeFontBoldItalic . . . 504
7.111.5.14 CodeFontItalic . . . 504
7.111.5.15 CodeInlineBackgroundColour . . . 504
7.111.5.16 CodeInlineMargin . . . 504
7.111.5.17 EmojiUriResolver . . . 505
7.111.5.18 EmojiURLMapping . . . 505
7.111.5.19 ForegroundColour . . . 505
7.111.5.20 HeaderFontSizeMultipliers . . . 505
7.111.5.21 HeaderLineColour . . . 506
7.111.5.22 HeaderLineThicknesses . . . 506
7.111.5.23 ImageMarginTolerance . . . 506
7.111.5.24 ImageMultiplier . . . 506
7.111.5.25 ImageSideMargin . . . 506
7.111.5.26 ImageUnitMultiplier . . . 507
7.111.5.27 ImageUriResolver . . . 507
7.111.5.28 IndentWidth . . . 507
7.111.5.29 InsertedColour . . . 507
7.111.5.30 ItalicFontFamily . . . 508
7.111.5.31 LinkColour . . . 508
7.111.5.32 LinkUriResolver . . . 508
7.111.5.33 Margins . . . 508
7.111.5.34 MarkdownPipelineBuilder . . . 508
7.111.5.35 MarkedColour . . . 509
7.111.5.36 MathFontScalingFactor . . . 509
7.111.5.37 PageSize . . . 509
7.111.5.38 QuoteBlockBackgroundColour . . . 509
7.111.5.39 QuoteBlockBarColour . . . 509
7.111.5.40 QuoteBlockBarWidth . . . 510
7.111.5.41 QuoteBlockIndentWidth . . . 510
7.111.5.42 RasterImageLoader . . . 510
7.111.5.43 RegularFontFamily . . . 510

7.111.5.44 SpaceAfterHeading . . . 510
7.111.5.45 SpaceAfterLine . . . 511
7.111.5.46 SpaceAfterParagraph . . . 511
7.111.5.47 SpaceBeforeHeading . . . 511
7.111.5.48 SpaceBeforeParagaph . . . 511
7.111.5.49 SubscriptShift . . . 511
7.111.5.50 SubSuperscriptFontSize . . . 512
7.111.5.51 SuperscriptShift . . . 512
7.111.5.52 SyntaxHighlighter . . . 512
7.111.5.53 TableCellMargins . . . 512
7.111.5.54 TableHeaderRowSeparatorColour . . . 513
7.111.5.55 TableHeaderRowSeparatorThickness . . . 513
7.111.5.56 TableHeaderSeparatorThickness . . . 513
7.111.5.57 TableRowSeparatorColour . . . 513
7.111.5.58 TableVAlign . . . 513
7.111.5.59 TaskListCheckedBullet . . . 514
7.111.5.60 TaskListUncheckedBullet . . . 514
7.111.5.61 ThematicBreakLineColour . . . 514
7.111.5.62 ThematicBreakThickness . . . 515
7.111.5.63 UnderlineThickness . . . 515
7.111.6 Event Documentation . . . 515
7.111.6.1 BlockRendered . . . 515
7.111.6.2 BlockRendering . . . 515
7.111.6.3 InlineRendered . . . 515
7.111.6.4 InlineRendering . . . 516
7.111.6.5 LineMeasured . . . 516
7.111.6.6 LineRendered . . . 516
7.111.6.7 LineRendering . . . 516
7.111.6.8 PageFinished . . . 516
7.111.6.9 PageStarted . . . 517
7.112 VectSharp.ThreeD.MaskedLightSource Class Reference . . . 517
7.112.1 Detailed Description . . . 518
7.112.2 Constructor & Destructor Documentation . . . 518
7.112.2.1 MaskedLightSource() [1/2] . . . 519
7.112.2.2 MaskedLightSource() [2/2] . . . 519
7.112.3 Member Function Documentation . . . 520
7.112.3.1 GetLightAt() . . . 520
7.112.3.2 GetObstruction() . . . 520
7.112.4 Property Documentation . . . 520
7.112.4.1 AngleAttenuationExponent . . . 521
7.112.4.2 CastsShadow . . . 521
7.112.4.3 Direction . . . 521

7.112.4.4 Distance . . . 521
7.112.4.5 DistanceAttenuationExponent . . . 521
7.112.4.6 Intensity . . . 522
7.112.4.7 Origin . . . 522
7.112.4.8 Position . . . 522
7.113 VectSharp.Filters.MaskFilter Class Reference . . . 522
7.113.1 Detailed Description . . . 523
7.113.2 Constructor & Destructor Documentation . . . 523
7.113.2.1 MaskFilter() . . . 523
7.113.3 Member Function Documentation . . . 523
7.113.3.1 Filter() . . . 524
7.113.4 Property Documentation . . . 524
7.113.4.1 BottomRightMargin . . . 524
7.113.4.2 Mask . . . 524
7.113.4.3 TopLeftMargin . . . 525
7.114 VectSharp.Plots.MovingAverageTrendLine Class Reference . . . 525
7.114.1 Detailed Description . . . 526
7.114.2 Constructor & Destructor Documentation . . . 526
7.114.2.1 MovingAverageTrendLine() [1/2] . . . 526
7.114.2.2 MovingAverageTrendLine() [2/2] . . . 527
7.114.3 Member Function Documentation . . . 527
7.114.3.1 Plot() . . . 527
7.114.4 Property Documentation . . . 527
7.114.4.1 CoordinateSystem . . . 528
7.114.4.2 Data . . . 528
7.114.4.3 Period . . . 528
7.114.4.4 PresentationAttributes . . . 528
7.114.4.5 Tag . . . 528
7.114.4.6 Weight . . . 529
7.115 VectSharp.MultiFontLibrary Class Reference . . . 529
7.115.1 Detailed Description . . . 530
7.115.2 Constructor & Destructor Documentation . . . 530
7.115.2.1 MultiFontLibrary() [1/2] . . . 530
7.115.2.2 MultiFontLibrary() [2/2] . . . 530
7.115.3 Member Function Documentation . . . 531
7.115.3.1 ResolveFontFamily() [1/2] . . . 531
7.115.3.2 ResolveFontFamily() [2/2] . . . 531
7.116 VectSharp.Fonts.Nimbus Class Reference . . . 532
7.116.1 Detailed Description . . . 532
7.116.2 Property Documentation . . . 532
7.116.2.1 Library . . . 532
7.117 VectSharp.ThreeD.ObjectFactory Class Reference . . . 533

7.117.1 Detailed Description . . . 533
7.117.2 Member Function Documentation . . . 533
7.117.2.1 CreateCube() . . . 534
7.117.2.2 CreateCuboid() . . . 534
7.117.2.3 CreatePoints() . . . 535
7.117.2.4 CreatePolygon() . . . 535
7.117.2.5 CreatePrism() . . . 536
7.117.2.6 CreateRectangle() [1/2] . . . 537
7.117.2.7 CreateRectangle() [2/2] . . . 537
7.117.2.8 CreateSphere() . . . 538
7.117.2.9 CreateTetrahedron() . . . 539
7.117.2.10 CreateWireframe() . . . 539
7.118 VectSharp.PDF.OptionalContentGroups.OptionalContentGroup Class Reference . . . 540
7.118.1 Detailed Description . . . 541
7.118.2 Member Enumeration Documentation . . . 541
7.118.2.1 Intents . . . 541
7.118.3 Constructor & Destructor Documentation . . . 541
7.118.3.1 OptionalContentGroup() . . . 541
7.118.4 Member Function Documentation . . . 542
7.118.4.1 CreateSetOCGLink() . . . 542
7.118.4.2 ToString() . . . 542
7.118.5 Property Documentation . . . 543
7.118.5.1 DefaultExportState . . . 543
7.118.5.2 DefaultPrintState . . . 543
7.118.5.3 DefaultState . . . 543
7.118.5.4 DefaultViewState . . . 543
7.118.5.5 Intent . . . 544
7.118.5.6 Name . . . 544
7.118.5.7 Operation . . . 544
7.119 VectSharp.PDF.OptionalContentGroups.OptionalContentGroupAndExpression Class Reference . . . 544
7.119.1 Detailed Description . . . 545
7.119.2 Constructor & Destructor Documentation . . . 545
7.119.2.1 OptionalContentGroupAndExpression() . . . 545
7.119.3 Property Documentation . . . 545
7.119.3.1 Operation . . . 546
7.120 VectSharp.PDF.OptionalContentGroups.OptionalContentGroupExpression Class Reference . . . 546
7.120.1 Detailed Description . . . 547
7.120.2 Member Function Documentation . . . 547
7.120.2.1 operator false() . . . 547
7.120.2.2 operator true() . . . 547
7.120.2.3 operator"!() . . . 548
7.120.2.4 operator&() . . . 548

7.120.2.5 operator"|() . . . 549
7.120.2.6 ToString() . . . 549
7.120.3 Property Documentation . . . 549
7.120.3.1 Arguments . . . 549
7.120.3.2 Operation . . . 550
7.121 VectSharp.PDF.OptionalContentGroups.OptionalContentGroupNotExpression Class Reference . . . 550
7.121.1 Detailed Description . . . 550
7.121.2 Constructor & Destructor Documentation . . . 551
7.121.2.1 OptionalContentGroupNotExpression() . . . 551
7.121.3 Property Documentation . . . 551
7.121.3.1 Operation . . . 551
7.122 VectSharp.PDF.OptionalContentGroups.OptionalContentGroupOrExpression Class Reference . . . 551
7.122.1 Detailed Description . . . 552
7.122.2 Constructor & Destructor Documentation . . . 552
7.122.2.1 OptionalContentGroupOrExpression() . . . 552
7.122.3 Property Documentation . . . 553
7.122.3.1 Operation . . . 553
7.123 VectSharp.PDF.OptionalContentGroups.OptionalContentGroupSettings Class Reference . . . 553
7.123.1 Detailed Description . . . 554
7.123.2 Constructor & Destructor Documentation . . . 554
7.123.2.1 OptionalContentGroupSettings() . . . 554
7.123.3 Property Documentation . . . 554
7.123.3.1 Groups . . . 554
7.123.3.2 OptionalContentGroupTree . . . 554
7.123.3.3 RadioButtonGroups . . . 555
7.124 VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeLabel Class Reference . . . 555
7.124.1 Detailed Description . . . 556
7.124.2 Member Enumeration Documentation . . . 556
7.124.2.1 Type . . . 556
7.124.3 Constructor & Destructor Documentation . . . 556
7.124.3.1 OptionalContentGroupTreeLabel() [1/2] . . . 556
7.124.3.2 OptionalContentGroupTreeLabel() [2/2] . . . 556
7.124.4 Member Function Documentation . . . 557
7.124.4.1 operator OptionalContentGroupTreeLabel() [1/2] . . . 557
7.124.4.2 operator OptionalContentGroupTreeLabel() [2/2] . . . 557
7.124.5 Property Documentation . . . 557
7.124.5.1 LabelOptionalContentGroup . . . 557
7.124.5.2 LabelString . . . 558
7.124.5.3 LabelType . . . 558
7.125 VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeNode Class Reference . . . 558
7.125.1 Detailed Description . . . 559
7.125.2 Constructor & Destructor Documentation . . . 559

7.125.2.1 OptionalContentGroupTreeNode() . . . 559
7.125.3 Member Function Documentation . . . 560
7.125.3.1 Add() . . . 560
7.125.3.2 Clear() . . . 560
7.125.3.3 Contains() . . . 560
7.125.3.4 CopyTo() . . . 560
7.125.3.5 GetEnumerator() . . . 561
7.125.3.6 IndexOf() . . . 561
7.125.3.7 Insert() . . . 561
7.125.3.8 operator OptionalContentGroupTreeNode() [1/3] . . . 561
7.125.3.9 operator OptionalContentGroupTreeNode() [2/3] . . . 561
7.125.3.10 operator OptionalContentGroupTreeNode() [3/3] . . . 562
7.125.3.11 Remove() . . . 562
7.125.3.12 RemoveAt() . . . 562
7.125.4 Property Documentation . . . 562
7.125.4.1 Children . . . 562
7.125.4.2 Count . . . 563
7.125.4.3 IsReadOnly . . . 563
7.125.4.4 Label . . . 563
7.125.4.5 this[int index] . . . 563
7.126 VectSharp.PDF.OutlineTree Class Reference . . . 563
7.126.1 Detailed Description . . . 564
7.126.2 Constructor & Destructor Documentation . . . 564
7.126.2.1 OutlineTree() . . . 564
7.126.3 Member Function Documentation . . . 564
7.126.3.1 Add() . . . 565
7.126.3.2 Clear() . . . 565
7.126.3.3 Contains() . . . 565
7.126.3.4 CopyTo() . . . 565
7.126.3.5 CreateFromHeadings() . . . 565
7.126.3.6 GetEnumerator() . . . 566
7.126.3.7 IndexOf() . . . 566
7.126.3.8 Insert() . . . 566
7.126.3.9 Remove() . . . 566
7.126.3.10 RemoveAt() . . . 566
7.126.4 Property Documentation . . . 567
7.126.4.1 Count . . . 567
7.126.4.2 IsReadOnly . . . 567
7.126.4.3 this[int index] . . . 567
7.126.4.4 TopLevelItems . . . 567
7.127 VectSharp.PDF.OutlineTreeNode Class Reference . . . 568
7.127.1 Detailed Description . . . 569

7.127.2 Constructor & Destructor Documentation . . . 569
7.127.2.1 OutlineTreeNode() . . . 569
7.127.3 Member Function Documentation . . . 569
7.127.3.1 Add() . . . 569
7.127.3.2 Clear() . . . 569
7.127.3.3 Contains() . . . 570
7.127.3.4 CopyTo() . . . 570
7.127.3.5 GetEnumerator() . . . 570
7.127.3.6 IndexOf() . . . 570
7.127.3.7 Insert() . . . 570
7.127.3.8 Remove() . . . 571
7.127.3.9 RemoveAt() . . . 571
7.127.4 Property Documentation . . . 571
7.127.4.1 Bold . . . 571
7.127.4.2 Children . . . 571
7.127.4.3 Colour . . . 571
7.127.4.4 Count . . . 572
7.127.4.5 DestinationTag . . . 572
7.127.4.6 IsReadOnly . . . 572
7.127.4.7 Italic . . . 572
7.127.4.8 this[int index] . . . 572
7.127.4.9 Title . . . 573
7.128 VectSharp.Page Class Reference . . . 573
7.128.1 Detailed Description . . . 573
7.128.2 Constructor & Destructor Documentation . . . 573
7.128.2.1 Page() . . . 573
7.128.3 Member Function Documentation . . . 574
7.128.3.1 Crop() [1/3] . . . 574
7.128.3.2 Crop() [2/3] . . . 574
7.128.3.3 Crop() [3/3] . . . 575
7.128.4 Property Documentation . . . 575
7.128.4.1 Background . . . 575
7.128.4.2 Graphics . . . 575
7.128.4.3 Height . . . 575
7.128.4.4 Width . . . 576
7.129 VectSharp.Markdown.PageFinishedEventArgs Class Reference . . . 576
7.129.1 Detailed Description . . . 577
7.129.2 Constructor & Destructor Documentation . . . 577
7.129.2.1 PageFinishedEventArgs() . . . 577
7.129.3 Property Documentation . . . 577
7.129.3.1 Graphics . . . 577
7.129.3.2 MarkdownContext . . . 577

7.129.3.3 Page . . . 578
7.130 VectSharp.Markdown.PageStartedEventArgs Class Reference . . . 578
7.130.1 Detailed Description . . . 579
7.130.2 Constructor & Destructor Documentation . . . 579
7.130.2.1 PageStartedEventArgs() . . . 579
7.130.3 Property Documentation . . . 579
7.130.3.1 Graphics . . . 579
7.130.3.2 MarkdownContext . . . 579
7.130.3.3 Page . . . 580
7.131 VectSharp.TrueTypeFile.PairKerning Class Reference . . . 580
7.131.1 Detailed Description . . . 580
7.131.2 Property Documentation . . . 580
7.131.2.1 Glyph1Advance . . . 580
7.131.2.2 Glyph1Placement . . . 581
7.131.2.3 Glyph2Advance . . . 581
7.131.2.4 Glyph2Placement . . . 581
7.132 VectSharp.ThreeD.ParallelLightSource Class Reference . . . 581
7.132.1 Detailed Description . . . 582
7.132.2 Constructor & Destructor Documentation . . . 582
7.132.2.1 ParallelLightSource() . . . 582
7.132.3 Member Function Documentation . . . 583
7.132.3.1 GetLightAt() . . . 583
7.132.3.2 GetObstruction() . . . 583
7.132.4 Property Documentation . . . 584
7.132.4.1 CastsShadow . . . 584
7.132.4.2 Direction . . . 584
7.132.4.3 Intensity . . . 584
7.132.4.4 ReverseDirection . . . 585
7.133 VectSharp.SVG.Parser Class Reference . . . 585
7.133.1 Detailed Description . . . 585
7.133.2 Member Function Documentation . . . 585
7.133.2.1 FromFile() . . . 585
7.133.2.2 FromStream() . . . 586
7.133.2.3 FromString() . . . 586
7.133.2.4 ParseSVGURI() . . . 587
7.133.3 Member Data Documentation . . . 587
7.133.3.1 ParseImageURI . . . 587
7.134 VectSharp.Plots.PathDataPointElement Class Reference . . . 588
7.134.1 Detailed Description . . . 588
7.134.2 Constructor & Destructor Documentation . . . 589
7.134.2.1 PathDataPointElement() [1/2] . . . 589
7.134.2.2 PathDataPointElement() [2/2] . . . 589

7.134.3 Member Function Documentation . . . 589
7.134.3.1 Plot() . . . 589
7.134.4 Property Documentation . . . 590
7.134.4.1 Path . . . 590
7.135 VectSharp.PDF.PDFContextInterpreter Class Reference . . . 590
7.135.1 Detailed Description . . . 591
7.135.2 Member Enumeration Documentation . . . 591
7.135.2.1 TextOptions . . . 591
7.135.3 Member Function Documentation . . . 591
7.135.3.1 CreatePDFDocument() . . . 591
7.135.3.2 SaveAsPDF() [1/2] . . . 592
7.135.3.3 SaveAsPDF() [2/2] . . . 592
7.136 VectSharp.PDF.PDFDocument Class Reference . . . 593
7.136.1 Detailed Description . . . 594
7.136.2 Constructor & Destructor Documentation . . . 594
7.136.2.1 PDFDocument() . . . 594
7.136.3 Member Function Documentation . . . 594
7.136.3.1 Write() [1/2] . . . 594
7.136.3.2 Write() [2/2] . . . 595
7.136.4 Member Data Documentation . . . 595
7.136.4.1 DefaultEncoding . . . 595
7.136.5 Property Documentation . . . 595
7.136.5.1 Catalog . . . 595
7.136.5.2 Contents . . . 596
7.136.5.3 Info . . . 596
7.136.5.4 Pages . . . 596
7.136.5.5 PDFVersion . . . 596
7.137 VectSharp.PDF.PDFMetadata Class Reference . . . 596
7.137.1 Detailed Description . . . 597
7.137.2 Constructor & Destructor Documentation . . . 597
7.137.2.1 PDFMetadata() . . . 598
7.137.3 Member Function Documentation . . . 598
7.137.3.1 NoMetadata() . . . 598
7.137.4 Property Documentation . . . 598
7.137.4.1 Author . . . 598
7.137.4.2 CreationDate . . . 598
7.137.4.3 CreationDateTimeZone . . . 599
7.137.4.4 Creator . . . 599
7.137.4.5 CustomProperties . . . 599
7.137.4.6 ExcludeMetadata . . . 599
7.137.4.7 Keywords . . . 599
7.137.4.8 ModificationDate . . . 600

7.137.4.9 ModificationDateTimeZone . . . 600
7.137.4.10 Producer . . . 600
7.137.4.11 Subject . . . 600
7.137.4.12 Title . . . 600
7.138 VectSharp.MuPDFUtils.PDFParser Class Reference . . . 601
7.138.1 Detailed Description . . . 601
7.138.2 Member Function Documentation . . . 601
7.138.2.1 FromFile() [1/4] . . . 601
7.138.2.2 FromFile() [2/4] . . . 602
7.138.2.3 FromFile() [3/4] . . . 602
7.138.2.4 FromFile() [4/4] . . . 603
7.138.2.5 FromStream() [1/4] . . . 604
7.138.2.6 FromStream() [2/4] . . . 604
7.138.2.7 FromStream() [3/4] . . . 605
7.138.2.8 FromStream() [4/4] . . . 605
7.139 VectSharp.ThreeD.PhongMaterial Class Reference . . . 606
7.139.1 Detailed Description . . . 607
7.139.2 Constructor & Destructor Documentation . . . 607
7.139.2.1 PhongMaterial() . . . 607
7.139.3 Member Function Documentation . . . 607
7.139.3.1 GetColour() . . . 608
7.139.4 Property Documentation . . . 608
7.139.4.1 AmbientReflectionCoefficient . . . 608
7.139.4.2 Colour . . . 608
7.139.4.3 DiffuseReflectionCoefficient . . . 609
7.139.4.4 SpecularReflectionCoefficient . . . 609
7.139.4.5 SpecularShininess . . . 609
7.140 VectSharp.Plots.Pie Class Reference . . . 609
7.140.1 Detailed Description . . . 610
7.140.2 Constructor & Destructor Documentation . . . 610
7.140.2.1 Pie() [1/2] . . . 611
7.140.2.2 Pie() [2/2] . . . 611
7.140.3 Member Function Documentation . . . 611
7.140.3.1 Plot() . . . 612
7.140.4 Property Documentation . . . 612
7.140.4.1 Centre . . . 612
7.140.4.2 Clockwise . . . 612
7.140.4.3 CoordinateSystem . . . 612
7.140.4.4 Data . . . 613
7.140.4.5 InnerRadius . . . 613
7.140.4.6 OuterRadius . . . 613
7.140.4.7 PresentationAttributes . . . 613

7.140.4.8 StartAngle . . . 613
7.140.4.9 Tag . . . 614
7.141 VectSharp.Plots.Plot Class Reference . . . 614
7.141.1 Detailed Description . . . 615
7.141.2 Member Enumeration Documentation . . . 615
7.141.2.1 NormalisationMode . . . 615
7.141.2.2 WhiskerType . . . 615
7.141.3 Constructor & Destructor Documentation . . . 615
7.141.3.1 Plot() . . . 615
7.141.4 Member Function Documentation . . . 615
7.141.4.1 AddPlotElement() . . . 615
7.141.4.2 AddPlotElements() [1/2] . . . 616
7.141.4.3 AddPlotElements() [2/2] . . . 616
7.141.4.4 GetAll< T >() . . . 616
7.141.4.5 GetFirst< T >() . . . 617
7.141.4.6 RemovePlotElement() . . . 617
7.141.4.7 Render() [1/2] . . . 618
7.141.4.8 Render() [2/2] . . . 618
7.141.5 Property Documentation . . . 618
7.141.5.1 PlotElements . . . 618
7.142 VectSharp.Plots.PlotElement< T > Class Template Reference . . . 619
7.142.1 Detailed Description . . . 619
7.142.2 Constructor & Destructor Documentation . . . 620
7.142.2.1 PlotElement() . . . 620
7.142.3 Member Function Documentation . . . 620
7.142.3.1 Plot() . . . 620
7.142.4 Property Documentation . . . 620
7.142.4.1 CoordinateSystem . . . 620
7.142.4.2 PlotAction . . . 621
7.143 VectSharp.Plots.PlotElementPresentationAttributes Class Reference . . . 621
7.143.1 Detailed Description . . . 622
7.143.2 Constructor & Destructor Documentation . . . 622
7.143.2.1 PlotElementPresentationAttributes() [1/2] . . . 622
7.143.2.2 PlotElementPresentationAttributes() [2/2] . . . 622
7.143.3 Property Documentation . . . 622
7.143.3.1 Fill . . . 622
7.143.3.2 Font . . . 623
7.143.3.3 LineCap . . . 623
7.143.3.4 LineDash . . . 623
7.143.3.5 LineJoin . . . 623
7.143.3.6 LineWidth . . . 623
7.143.3.7 Stroke . . . 624

7.144 VectSharp.Point Struct Reference . . . 624
7.144.1 Detailed Description . . . 625
7.144.2 Constructor & Destructor Documentation . . . 625
7.144.2.1 Point() . . . 626
7.144.3 Member Function Documentation . . . 627
7.144.3.1 Bounds() [1/2] . . . 627
7.144.3.2 Bounds() [2/2] . . . 627
7.144.3.3 GetEnumerator() . . . 628
7.144.3.4 IsEqual() . . . 628
7.144.3.5 Max() . . . 628
7.144.3.6 Min() . . . 629
7.144.3.7 Modulus() . . . 629
7.144.3.8 Normalize() . . . 629
7.144.3.9 operator() . . . 629
7.144.3.10 operator double[]() . . . 630
7.144.3.11 operator Point() [1/2] . . . 630
7.144.3.12 operator Point() [2/2] . . . 630
7.144.3.13 operator∗() [1/2] . . . 631
7.144.3.14 operator∗() [2/2] . . . 631
7.144.3.15 operator+() . . . 631
7.144.3.16 operator-() . . . 632
7.144.4 Member Data Documentation . . . 632
7.144.4.1 X . . . 632
7.144.4.2 Y . . . 633
7.144.5 Property Documentation . . . 633
7.144.5.1 Count . . . 633
7.144.5.2 this[int index] . . . 633
7.145 VectSharp.ThreeD.PointLightSource Class Reference . . . 633
7.145.1 Detailed Description . . . 634
7.145.2 Constructor & Destructor Documentation . . . 634
7.145.2.1 PointLightSource() . . . 634
7.145.3 Member Function Documentation . . . 635
7.145.3.1 GetLightAt() . . . 635
7.145.3.2 GetObstruction() . . . 635
7.145.4 Property Documentation . . . 636
7.145.4.1 CastsShadow . . . 636
7.145.4.2 DistanceAttenuationExponent . . . 636
7.145.4.3 Intensity . . . 636
7.145.4.4 Position . . . 637
7.146 VectSharp.Plots.PolynomialTrendLine Class Reference . . . 637
7.146.1 Detailed Description . . . 638
7.146.2 Constructor & Destructor Documentation . . . 638

7.146.2.1 PolynomialTrendLine() [1/3] . . . 638
7.146.2.2 PolynomialTrendLine() [2/3] . . . 639
7.146.2.3 PolynomialTrendLine() [3/3] . . . 639
7.146.3 Member Function Documentation . . . 640
7.146.3.1 Plot() . . . 640
7.146.4 Property Documentation . . . 640
7.146.4.1 Coefficients . . . 640
7.146.4.2 CoordinateSystem . . . 640
7.146.4.3 MaxX . . . 641
7.146.4.4 MaxY . . . 641
7.146.4.5 MinX . . . 641
7.146.4.6 MinY . . . 641
7.146.4.7 PresentationAttributes . . . 641
7.146.4.8 Tag . . . 642
7.147 VectSharp.Plots.PowerLawTrendLine Class Reference . . . 642
7.147.1 Detailed Description . . . 643
7.147.2 Constructor & Destructor Documentation . . . 643
7.147.2.1 PowerLawTrendLine() [1/3] . . . 643
7.147.2.2 PowerLawTrendLine() [2/3] . . . 644
7.147.2.3 PowerLawTrendLine() [3/3] . . . 644
7.147.3 Member Function Documentation . . . 644
7.147.3.1 Plot() . . . 644
7.147.4 Property Documentation . . . 645
7.147.4.1 CoordinateSystem . . . 645
7.147.4.2 Intercept . . . 645
7.147.4.3 MaxX . . . 645
7.147.4.4 MaxY . . . 646
7.147.4.5 MinX . . . 646
7.147.4.6 MinY . . . 646
7.147.4.7 PresentationAttributes . . . 646
7.147.4.8 Slope . . . 646
7.147.4.9 Tag . . . 647
7.148 VectSharp.RadialGradientBrush Class Reference . . . 647
7.148.1 Detailed Description . . . 648
7.148.2 Constructor & Destructor Documentation . . . 648
7.148.2.1 RadialGradientBrush() [1/2] . . . 648
7.148.2.2 RadialGradientBrush() [2/2] . . . 649
7.148.3 Member Function Documentation . . . 649
7.148.3.1 MultiplyOpacity() . . . 649
7.148.4 Property Documentation . . . 649
7.148.4.1 Centre . . . 650
7.148.4.2 FocalPoint . . . 650

7.148.4.3 Radius . . . 650
7.149 VectSharp.TrueTypeFile.CoverageTable.RangeRecord Struct Reference . . . 650
7.149.1 Detailed Description . . . 650
7.149.2 Constructor & Destructor Documentation . . . 651
7.149.2.1 RangeRecord() . . . 651
7.149.3 Member Data Documentation . . . 651
7.149.3.1 EndGlyphID . . . 651
7.149.3.2 StartCoverageIndex . . . 651
7.149.3.3 StartGlyphID . . . 651
7.150 VectSharp.Raster.Raster Class Reference . . . 651
7.150.1 Detailed Description . . . 652
7.150.2 Member Function Documentation . . . 652
7.150.2.1 Rasterise() . . . 652
7.150.2.2 SaveAsAnimatedPNG() [1/2] . . . 653
7.150.2.3 SaveAsAnimatedPNG() [2/2] . . . 653
7.150.2.4 SaveAsJPEG() . . . 654
7.150.2.5 SaveAsPNG() [1/2] . . . 654
7.150.2.6 SaveAsPNG() [2/2] . . . 655
7.150.2.7 SaveAsRawBytes() . . . 655
7.151 VectSharp.RasterImage Class Reference . . . 656
7.151.1 Detailed Description . . . 657
7.151.2 Constructor & Destructor Documentation . . . 657
7.151.2.1 RasterImage() [1/3] . . . 657
7.151.2.2 RasterImage() [2/3] . . . 658
7.151.2.3 RasterImage() [3/3] . . . 658
7.151.3 Member Function Documentation . . . 659
7.151.3.1 ClearPNGCache() . . . 659
7.151.3.2 Dispose() . . . 659
7.151.4 Property Documentation . . . 659
7.151.4.1 DataHolder . . . 659
7.151.4.2 HasAlpha . . . 659
7.151.4.3 Height . . . 660
7.151.4.4 Id . . . 660
7.151.4.5 ImageDataAddress . . . 660
7.151.4.6 Interpolate . . . 660
7.151.4.7 PNGStream . . . 660
7.151.4.8 Width . . . 661
7.152 VectSharp.ImageSharpUtils.RasterImageFile Class Reference . . . 661
7.152.1 Detailed Description . . . 661
7.152.2 Constructor & Destructor Documentation . . . 662
7.152.2.1 RasterImageFile() . . . 662
7.153 VectSharp.MuPDFUtils.RasterImageFile Class Reference . . . 662

7.153.1 Detailed Description . . . 663
7.153.2 Constructor & Destructor Documentation . . . 663
7.153.2.1 RasterImageFile() . . . 663
7.154 VectSharp.ImageSharpUtils.RasterImageStream Class Reference . . . 663
7.154.1 Detailed Description . . . 664
7.154.2 Constructor & Destructor Documentation . . . 664
7.154.2.1 RasterImageStream() [1/2] . . . 664
7.154.2.2 RasterImageStream() [2/2] . . . 665
7.155 VectSharp.MuPDFUtils.RasterImageStream Class Reference . . . 665
7.155.1 Detailed Description . . . 666
7.155.2 Constructor & Destructor Documentation . . . 666
7.155.2.1 RasterImageStream() [1/2] . . . 666
7.155.2.2 RasterImageStream() [2/2] . . . 667
7.156 VectSharp.Rectangle Struct Reference . . . 667
7.156.1 Detailed Description . . . 668
7.156.2 Constructor & Destructor Documentation . . . 669
7.156.2.1 Rectangle() [1/3] . . . 669
7.156.2.2 Rectangle() [2/3] . . . 670
7.156.2.3 Rectangle() [3/3] . . . 670
7.156.3 Member Function Documentation . . . 670
7.156.3.1 Intersection() . . . 671
7.156.3.2 IsNaN() . . . 671
7.156.3.3 Union() [1/3] . . . 671
7.156.3.4 Union() [2/3] . . . 672
7.156.3.5 Union() [3/3] . . . 672
7.156.4 Member Data Documentation . . . 672
7.156.4.1 Location . . . 673
7.156.4.2 NaN . . . 673
7.156.4.3 Size . . . 673
7.156.5 Property Documentation . . . 673
7.156.5.1 Centre . . . 673
7.157 VectSharp.Markdown.RectangleFragment Class Reference . . . 674
7.157.1 Detailed Description . . . 675
7.157.2 Constructor & Destructor Documentation . . . 675
7.157.2.1 RectangleFragment() . . . 675
7.157.3 Member Function Documentation . . . 676
7.157.3.1 GetAscent() . . . 676
7.157.3.2 GetMaxX() . . . 676
7.157.3.3 GetMaxY() . . . 676
7.157.3.4 Render() . . . 677
7.157.3.5 Translate() . . . 677
7.157.4 Property Documentation . . . 677

7.157.4.1 Colour . . . 677
7.157.4.2 Size . . . 678
7.157.4.3 TopLeft . . . 678
7.158 VectSharp.Canvas.RenderAction Class Reference . . . 678
7.158.1 Detailed Description . . . 679
7.158.2 Member Enumeration Documentation . . . 680
7.158.2.1 ActionTypes . . . 680
7.158.3 Member Function Documentation . . . 680
7.158.3.1 BringToFront() . . . 680
7.158.3.2 ImageAction() . . . 680
7.158.3.3 PathAction() . . . 681
7.158.3.4 SendToBack() . . . 681
7.158.3.5 TextAction() . . . 682
7.158.4 Property Documentation . . . 682
7.158.4.1 ActionType . . . 682
7.158.4.2 ClippingPath . . . 682
7.158.4.3 Fill . . . 683
7.158.4.4 Geometry . . . 683
7.158.4.5 ImageDestination . . . 683
7.158.4.6 ImageId . . . 683
7.158.4.7 ImageSource . . . 683
7.158.4.8 InverseTransform . . . 684
7.158.4.9 Layout . . . 684
7.158.4.10 Parent . . . 684
7.158.4.11 Stroke . . . 684
7.158.4.12 Tag . . . 684
7.158.4.13 Text . . . 685
7.158.4.14 Transform . . . 685
7.158.5 Event Documentation . . . 685
7.158.5.1 PointerEntered . . . 685
7.158.5.2 PointerExited . . . 685
7.158.5.3 PointerPressed . . . 685
7.158.5.4 PointerReleased . . . 686
7.159 VectSharp.ResourceFontFamily Class Reference . . . 686
7.159.1 Detailed Description . . . 686
7.159.2 Constructor & Destructor Documentation . . . 687
7.159.2.1 ResourceFontFamily() . . . 687
7.159.3 Member Data Documentation . . . 687
7.159.3.1 ResourceName . . . 687
7.160 VectSharp.Plots.ScatterPoints< T > Class Template Reference . . . 687
7.160.1 Detailed Description . . . 688
7.160.2 Constructor & Destructor Documentation . . . 688

7.160.2.1 ScatterPoints() . . . 688
7.160.3 Member Function Documentation . . . 689
7.160.3.1 Plot() . . . 689
7.160.4 Property Documentation . . . 689
7.160.4.1 CoordinateSystem . . . 689
7.160.4.2 Data . . . 689
7.160.4.3 DataPointElement . . . 690
7.160.4.4 PresentationAttributes . . . 690
7.160.4.5 Size . . . 690
7.160.4.6 Tag . . . 690
7.161 VectSharp.ThreeD.Scene Class Reference . . . 691
7.161.1 Detailed Description . . . 692
7.161.2 Constructor & Destructor Documentation . . . 692
7.161.2.1 Scene() . . . 692
7.161.3 Member Function Documentation . . . 692
7.161.3.1 AddElement() . . . 692
7.161.3.2 AddRange() . . . 693
7.161.3.3 Replace() [1/2] . . . 693
7.161.3.4 Replace() [2/2] . . . 693
7.161.4 Property Documentation . . . 694
7.161.4.1 SceneElements . . . 694
7.161.4.2 SceneLock . . . 694
7.162 VectSharp.Segment Class Reference . . . 694
7.162.1 Detailed Description . . . 695
7.162.2 Member Function Documentation . . . 695
7.162.2.1 Clone() . . . 695
7.162.2.2 Flatten() . . . 695
7.162.2.3 FlattenForOffsetAndGetTangents() . . . 696
7.162.2.4 GetLinearisationTangents() . . . 696
7.162.2.5 GetPointAt() . . . 697
7.162.2.6 GetTangentAt() . . . 697
7.162.2.7 Linearise() . . . 697
7.162.2.8 Measure() . . . 698
7.162.2.9 Transform() . . . 698
7.162.3 Property Documentation . . . 698
7.162.3.1 Point . . . 698
7.162.3.2 Points . . . 699
7.162.3.3 Type . . . 699
7.163 VectSharp.SimpleFontLibrary Class Reference . . . 699
7.163.1 Detailed Description . . . 700
7.163.2 Constructor & Destructor Documentation . . . 700
7.163.2.1 SimpleFontLibrary() [1/4] . . . 701

7.163.2.2 SimpleFontLibrary() [2/4] . . . 701
7.163.2.3 SimpleFontLibrary() [3/4] . . . 701
7.163.2.4 SimpleFontLibrary() [4/4] . . . 702
7.163.3 Member Function Documentation . . . 703
7.163.3.1 Add() [1/4] . . . 703
7.163.3.2 Add() [2/4] . . . 703
7.163.3.3 Add() [3/4] . . . 703
7.163.3.4 Add() [4/4] . . . 704
7.163.3.5 ResolveFontFamily() [1/2] . . . 704
7.163.3.6 ResolveFontFamily() [2/2] . . . 704
7.164 VectSharp.Size Struct Reference . . . 705
7.164.1 Detailed Description . . . 705
7.164.2 Constructor & Destructor Documentation . . . 705
7.164.2.1 Size() . . . 705
7.164.3 Member Data Documentation . . . 706
7.164.3.1 Height . . . 706
7.164.3.2 Width . . . 706
7.165 VectSharp.Canvas.SKMultiLayerRenderCanvas Class Reference . . . 706
7.165.1 Detailed Description . . . 708
7.165.2 Constructor & Destructor Documentation . . . 708
7.165.2.1 SKMultiLayerRenderCanvas() [1/3] . . . 708
7.165.2.2 SKMultiLayerRenderCanvas() [2/3] . . . 709
7.165.2.3 SKMultiLayerRenderCanvas() [3/3] . . . 709
7.165.3 Member Function Documentation . . . 710
7.165.3.1 AddLayer() . . . 710
7.165.3.2 Dispose() . . . 710
7.165.3.3 InsertLayer() . . . 710
7.165.3.4 InvalidateDirty() . . . 711
7.165.3.5 InvalidateZIndex() . . . 711
7.165.3.6 MoveLayer() . . . 711
7.165.3.7 RemoveLayer() . . . 711
7.165.3.8 Render() . . . 712
7.165.3.9 RenderAtResolution() . . . 712
7.165.3.10 SwitchLayers() . . . 712
7.165.3.11 UpdateLayer() . . . 713
7.165.3.12 UpdateWith() . . . 713
7.165.4 Member Data Documentation . . . 714
7.165.4.1 LayerTransforms . . . 714
7.165.4.2 RenderActions . . . 714
7.165.4.3 RenderLock . . . 714
7.165.5 Property Documentation . . . 714
7.165.5.1 ClipMargin . . . 714

7.165.5.2 PageHeight . . . 715
7.165.5.3 PageWidth . . . 715
7.166 VectSharp.Canvas.SKRenderAction Class Reference . . . 715
7.166.1 Detailed Description . . . 717
7.166.2 Member Enumeration Documentation . . . 717
7.166.2.1 ActionTypes . . . 718
7.166.3 Member Function Documentation . . . 718
7.166.3.1 ClipAction() . . . 718
7.166.3.2 Dispose() . . . 718
7.166.3.3 DrawFilteredGraphicsAction() . . . 718
7.166.3.4 ImageAction() . . . 719
7.166.3.5 InvalidateAll() . . . 719
7.166.3.6 InvalidateHitTestPath() . . . 719
7.166.3.7 InvalidateVisual() . . . 720
7.166.3.8 InvalidateZIndex() . . . 720
7.166.3.9 PathAction() . . . 720
7.166.3.10 RestoreAction() . . . 721
7.166.3.11 SaveAction() . . . 721
7.166.3.12 TextAction() . . . 721
7.166.3.13 TransformAction() . . . 722
7.166.4 Property Documentation . . . 722
7.166.4.1 ActionType . . . 723
7.166.4.2 Disposed . . . 723
7.166.4.3 Filter . . . 723
7.166.4.4 Font . . . 723
7.166.4.5 Graphics . . . 723
7.166.4.6 ImageDestination . . . 724
7.166.4.7 ImageId . . . 724
7.166.4.8 ImageSource . . . 724
7.166.4.9 Paint . . . 724
7.166.4.10 Parent . . . 724
7.166.4.11 Path . . . 725
7.166.4.12 Payload . . . 725
7.166.4.13 Tag . . . 725
7.166.4.14 Text . . . 725
7.166.4.15 TextX . . . 725
7.166.4.16 TextY . . . 726
7.166.4.17 Transform . . . 726
7.166.4.18 ZIndex . . . 726
7.166.5 Event Documentation . . . 726
7.166.5.1 PointerEntered . . . 726
7.166.5.2 PointerExited . . . 727

7.166.5.3 PointerPressed . . . 727
7.166.5.4 PointerReleased . . . 727
7.167 VectSharp.Canvas.SKRenderContext Class Reference . . . 727
7.167.1 Detailed Description . . . 727
7.168 VectSharp.Canvas.SKRenderContextInterpreter Class Reference . . . 728
7.168.1 Detailed Description . . . 729
7.168.2 Member Function Documentation . . . 729
7.168.2.1 CopyToSKRenderContext() [1/3] . . . 729
7.168.2.2 CopyToSKRenderContext() [2/3] . . . 729
7.168.2.3 CopyToSKRenderContext() [3/3] . . . 730
7.168.2.4 PaintToSKCanvas() [1/6] . . . 731
7.168.2.5 PaintToSKCanvas() [2/6] . . . 732
7.168.2.6 PaintToSKCanvas() [3/6] . . . 733
7.168.2.7 PaintToSKCanvas() [4/6] . . . 733
7.168.2.8 PaintToSKCanvas() [5/6] . . . 734
7.168.2.9 PaintToSKCanvas() [6/6] . . . 734
7.168.2.10 Rasterise() . . . 736
7.169 VectSharp.SolidColourBrush Class Reference . . . 737
7.169.1 Detailed Description . . . 738
7.169.2 Constructor & Destructor Documentation . . . 738
7.169.2.1 SolidColourBrush() . . . 738
7.169.3 Member Function Documentation . . . 738
7.169.3.1 MultiplyOpacity() . . . 738
7.169.3.2 operator SolidColourBrush() . . . 739
7.169.4 Property Documentation . . . 739
7.169.4.1 A . . . 739
7.169.4.2 B . . . 739
7.169.4.3 Colour . . . 740
7.169.4.4 G . . . 740
7.169.4.5 R . . . 740
7.170 VectSharp.SplineEasing Class Reference . . . 740
7.170.1 Detailed Description . . . 741
7.170.2 Constructor & Destructor Documentation . . . 741
7.170.2.1 SplineEasing() . . . 741
7.170.3 Member Function Documentation . . . 741
7.170.3.1 Ease() . . . 742
7.170.4 Property Documentation . . . 742
7.170.4.1 ControlPoint1 . . . 742
7.170.4.2 ControlPoint2 . . . 742
7.171 VectSharp.ThreeD.SpotlightLightSource Class Reference . . . 743
7.171.1 Detailed Description . . . 744
7.171.2 Constructor & Destructor Documentation . . . 744

7.171.2.1 SpotlightLightSource() . . . 744
7.171.3 Member Function Documentation . . . 745
7.171.3.1 GetLightAt() . . . 745
7.171.3.2 GetObstruction() . . . 745
7.171.4 Property Documentation . . . 745
7.171.4.1 AngleAttenuationExponent . . . 746
7.171.4.2 BeamWidthAngle . . . 746
7.171.4.3 CastsShadow . . . 746
7.171.4.4 CutoffAngle . . . 746
7.171.4.5 Direction . . . 746
7.171.4.6 DistanceAttenuationExponent . . . 747
7.171.4.7 Intensity . . . 747
7.171.4.8 Position . . . 747
7.172 VectSharp.Plots.StackedBars Class Reference . . . 747
7.172.1 Detailed Description . . . 748
7.172.2 Constructor & Destructor Documentation . . . 748
7.172.2.1 StackedBars() [1/3] . . . 749
7.172.2.2 StackedBars() [2/3] . . . 749
7.172.2.3 StackedBars() [3/3] . . . 750
7.172.3 Member Function Documentation . . . 750
7.172.3.1 Plot() . . . 750
7.172.4 Property Documentation . . . 750
7.172.4.1 CoordinateSystem . . . 750
7.172.4.2 Data . . . 751
7.172.4.3 GetBaseline . . . 751
7.172.4.4 Margin . . . 751
7.172.4.5 PresentationAttributes . . . 751
7.172.4.6 Tag . . . 752
7.172.4.7 Vertical . . . 752
7.173 VectSharp.SVG.SVGContextInterpreter Class Reference . . . 752
7.173.1 Detailed Description . . . 753
7.173.2 Member Enumeration Documentation . . . 753
7.173.2.1 TextOptions . . . 754
7.173.3 Member Function Documentation . . . 754
7.173.3.1 SaveAsAnimatedSVG() [1/3] . . . 754
7.173.3.2 SaveAsAnimatedSVG() [2/3] . . . 754
7.173.3.3 SaveAsAnimatedSVG() [3/3] . . . 755
7.173.3.4 SaveAsAnimatedSVGWithFrames() [1/3] . . . 756
7.173.3.5 SaveAsAnimatedSVGWithFrames() [2/3] . . . 756
7.173.3.6 SaveAsAnimatedSVGWithFrames() [3/3] . . . 757
7.173.3.7 SaveAsSVG() [1/3] . . . 758
7.173.3.8 SaveAsSVG() [2/3] . . . 758

7.173.3.9 SaveAsSVG() [3/3] . . . 759
7.173.3.10 WriteSVGXML() [1/3] . . . 759
7.173.3.11 WriteSVGXML() [2/3] . . . 760
7.173.3.12 WriteSVGXML() [3/3] . . . 760
7.174 VectSharp.Plots.Swarm Class Reference . . . 761
7.174.1 Detailed Description . . . 762
7.174.2 Constructor & Destructor Documentation . . . 762
7.174.2.1 Swarm() . . . 762
7.174.3 Member Function Documentation . . . 762
7.174.3.1 Plot() . . . 762
7.174.4 Property Documentation . . . 763
7.174.4.1 CoordinateSystem . . . 763
7.174.4.2 Data . . . 763
7.174.4.3 Direction . . . 763
7.174.4.4 PointMargin . . . 764
7.174.4.5 PointSize . . . 764
7.174.4.6 Position . . . 764
7.174.4.7 PresentationAttributes . . . 764
7.174.4.8 SwarmPointElement . . . 764
7.174.4.9 Tag . . . 765
7.175 VectSharp.Markdown.SyntaxHighlighter Class Reference . . . 765
7.175.1 Detailed Description . . . 765
7.175.2 Member Function Documentation . . . 765
7.175.2.1 GetSyntaxHighlightedLines() . . . 765
7.176 VectSharp.Markdown.TextFragment Class Reference . . . 766
7.176.1 Detailed Description . . . 767
7.176.2 Constructor & Destructor Documentation . . . 767
7.176.2.1 TextFragment() . . . 767
7.176.3 Member Function Documentation . . . 768
7.176.3.1 GetAscent() . . . 768
7.176.3.2 GetMaxX() . . . 768
7.176.3.3 GetMaxY() . . . 769
7.176.3.4 Render() . . . 769
7.176.3.5 Translate() . . . 769
7.176.4 Property Documentation . . . 770
7.176.4.1 Colour . . . 770
7.176.4.2 Font . . . 770
7.176.4.3 Position . . . 770
7.176.4.4 Text . . . 770
7.177 VectSharp.Plots.TextLabel< T > Class Template Reference . . . 771
7.177.1 Detailed Description . . . 772
7.177.2 Constructor & Destructor Documentation . . . 772

7.177.2.1 TextLabel() . . . 772
7.177.3 Member Function Documentation . . . 772
7.177.3.1 Plot() . . . 772
7.177.4 Property Documentation . . . 773
7.177.4.1 Alignment . . . 773
7.177.4.2 Baseline . . . 773
7.177.4.3 CoordinateSystem . . . 773
7.177.4.4 Label . . . 773
7.177.4.5 Position . . . 774
7.177.4.6 PresentationAttributes . . . 774
7.177.4.7 Rotation . . . 774
7.177.4.8 Tag . . . 774
7.178 VectSharp.TextSpacing Struct Reference . . . 775
7.178.1 Detailed Description . . . 776
7.178.2 Constructor & Destructor Documentation . . . 776
7.178.2.1 TextSpacing() [1/2] . . . 776
7.178.2.2 TextSpacing() [2/2] . . . 776
7.178.3 Member Function Documentation . . . 777
7.178.3.1 Equals() [1/2] . . . 777
7.178.3.2 Equals() [2/2] . . . 777
7.178.3.3 GetHashCode() . . . 777
7.178.3.4 operator"!=() . . . 777
7.178.3.5 operator==() . . . 778
7.178.4 Property Documentation . . . 778
7.178.4.1 Default . . . 778
7.178.4.2 NonWhitespaceAdd . . . 778
7.178.4.3 NonWhitespaceScale . . . 778
7.178.4.4 WhitespaceAdd . . . 779
7.178.4.5 WhitespaceScale . . . 779
7.179 VectSharp.Transition Class Reference . . . 779
7.179.1 Detailed Description . . . 779
7.179.2 Constructor & Destructor Documentation . . . 780
7.179.2.1 Transition() . . . 780
7.179.3 Property Documentation . . . 781
7.179.3.1 Duration . . . 781
7.179.3.2 Easings . . . 781
7.179.3.3 OverallEasing . . . 781
7.180 VectSharp.TrueTypeFile Class Reference . . . 782
7.180.1 Detailed Description . . . 784
7.180.2 Member Function Documentation . . . 784
7.180.2.1 Destroy() . . . 784
7.180.2.2 Get1000EmAscent() . . . 784

7.180.2.3 Get1000EmDescent() . . . 784
7.180.2.4 Get1000EmGlyphBearings() . . . 784
7.180.2.5 Get1000EmGlyphVerticalMetrics() . . . 785
7.180.2.6 Get1000EmGlyphWidth() [1/2] . . . 785
7.180.2.7 Get1000EmGlyphWidth() [2/2] . . . 786
7.180.2.8 Get1000EmKerning() [1/2] . . . 786
7.180.2.9 Get1000EmKerning() [2/2] . . . 786
7.180.2.10 Get1000EmUnderlineIntersections() . . . 787
7.180.2.11 Get1000EmUnderlinePosition() . . . 787
7.180.2.12 Get1000EmUnderlineThickness() . . . 788
7.180.2.13 Get1000EmWinAscent() . . . 788
7.180.2.14 Get1000EmXMax() . . . 788
7.180.2.15 Get1000EmXMin() . . . 788
7.180.2.16 Get1000EmYMax() . . . 789
7.180.2.17 Get1000EmYMin() . . . 789
7.180.2.18 GetFirstCharIndex() . . . 789
7.180.2.19 GetFontFamilyName() . . . 789
7.180.2.20 GetFontName() . . . 790
7.180.2.21 GetFullFontFamilyName() . . . 790
7.180.2.22 GetGlyphIndex() . . . 790
7.180.2.23 GetGlyphPath() [1/2] . . . 791
7.180.2.24 GetGlyphPath() [2/2] . . . 791
7.180.2.25 GetItalicAngle() . . . 791
7.180.2.26 GetLastCharIndex() . . . 792
7.180.2.27 GetNames() [1/2] . . . 792
7.180.2.28 GetNames() [2/2] . . . 792
7.180.2.29 GetUnitsPerEm() . . . 793
7.180.2.30 IsBold() . . . 793
7.180.2.31 IsFixedPitch() . . . 793
7.180.2.32 IsItalic() . . . 794
7.180.2.33 IsOblique() . . . 794
7.180.2.34 IsScript() . . . 794
7.180.2.35 IsSerif() . . . 794
7.180.2.36 SubsetFont() . . . 794
7.180.3 Property Documentation . . . 795
7.180.3.1 FontStream . . . 795
7.181 VectSharp.TrueTypeFile.TrueTypeName Struct Reference . . . 795
7.181.1 Detailed Description . . . 796
7.181.2 Member Enumeration Documentation . . . 796
7.181.2.1 NameIdentifier . . . 796
7.181.3 Member Data Documentation . . . 796
7.181.3.1 Name . . . 796

7.181.3.2 NameId . . . 796
7.182 VectSharp.TrueTypeFile.TrueTypePoint Struct Reference . . . 796
7.182.1 Detailed Description . . . 797
7.182.2 Member Data Documentation . . . 797
7.182.2.1 IsOnCurve . . . 797
7.182.2.2 X . . . 797
7.182.2.3 Y . . . 797
7.183 VectSharp.UnbalancedStackException Class Reference . . . 798
7.183.1 Detailed Description . . . 798
7.184 VectSharp.Markdown.UnderlineFragment Class Reference . . . 798
7.184.1 Detailed Description . . . 800
7.184.2 Constructor & Destructor Documentation . . . 800
7.184.2.1 UnderlineFragment() . . . 800
7.184.3 Member Function Documentation . . . 800
7.184.3.1 GetAscent() . . . 800
7.184.3.2 GetMaxX() . . . 801
7.184.3.3 GetMaxY() . . . 801
7.184.3.4 Render() . . . 801
7.184.3.5 Translate() . . . 802
7.184.4 Property Documentation . . . 802
7.184.4.1 Colour . . . 802
7.184.4.2 End . . . 803
7.184.4.3 Start . . . 803
7.184.4.4 Thickness . . . 803
7.185 VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.UnknownFormatException Class Reference . . . 803
7.185.1 Detailed Description . . . 804
7.185.2 Property Documentation . . . 804
7.185.2.1 Format . . . 804
7.186 VectSharp.TrueTypeFile.VerticalMetrics Struct Reference . . . 804
7.186.1 Detailed Description . . . 804
7.186.2 Member Data Documentation . . . 805
7.186.2.1 YMax . . . 805
7.186.2.2 YMin . . . 805
7.187 VectSharp.Plots.Violin Class Reference . . . 805
7.187.1 Detailed Description . . . 806
7.187.2 Member Enumeration Documentation . . . 806
7.187.2.1 ViolinSide . . . 806
7.187.3 Constructor & Destructor Documentation . . . 807
7.187.3.1 Violin() . . . 807
7.187.4 Member Function Documentation . . . 807
7.187.4.1 Plot() . . . 807

7.187.5 Property Documentation . . . 807
7.187.5.1 CoordinateSystem . . . 808
7.187.5.2 Data . . . 808
7.187.5.3 Direction . . . 808
7.187.5.4 MaxBins . . . 808
7.187.5.5 Position . . . 808
7.187.5.6 PresentationAttributes . . . 809
7.187.5.7 Sides . . . 809
7.187.5.8 Smooth . . . 809
7.187.5.9 Tag . . . 809
7.187.5.10 Width . . . 809

8 File Documentation **811**

8.1 AnimatedCanvas.cs . . . 811
8.2 AvaloniaContext.cs . . . 815
8.3 RenderingParameters.cs . . . 850
8.4 SkiaBitmap.cs . . . 852
8.5 SKMultiLayerRenderCanvas.cs . . . 853
8.6 SKRenderContext.cs . . . 868
8.7 Nimbus.cs . . . 892
8.8 HtmlTag.cs . . . 893
8.9 Line.cs . . . 899
8.10 MarkdownContext.cs . . . 905
8.11 MarkdownRenderer.Blocks.cs . . . 909
8.12 MarkdownRenderer.cs . . . 919
8.13 MarkdownRenderer.Events.cs . . . 926
8.14 MarkdownRenderer.Inlines.cs . . . 933
8.15 MarkdownRenderer.Properties.cs . . . 943
8.16 MarkdownRenderer.Tables.cs . . . 949
8.17 SyntaxHighlighting.cs . . . 953
8.18 UnicodeEmojiParser.cs . . . 956
8.19 Word.cs . . . 958
8.20 ImageCache.cs . . . 960
8.21 MarkdownCanvas.axaml.cs . . . 961
8.22 ImageUriParser.cs . . . 965
8.23 ImageURIParser.cs . . . 966
8.24 PDFParser.cs . . . 969
8.25 RasterImages.cs . . . 974
8.26 RasterImages.cs . . . 977
8.27 AnnotationStyles.cs . . . 979
8.28 Figures.cs . . . 981
8.29 OptionalContentGroups.cs . . . 986

8.30 OutlineTree.cs . . . 993
8.31 PDFContext.cs . . . 996
8.32 PDFContextInterpreter.cs . . . 1006
8.33 PDFDocument.cs . . . 1018
8.34 PDFMetadata.cs . . . 1022
8.35 PDFProcessFigure.cs . . . 1024
8.36 Axes.cs . . . 1032
8.37 Bars.cs . . . 1044
8.38 BoxPlot.cs . . . 1058
8.39 CoordinateSystems.cs . . . 1062
8.40 DataPoints.cs . . . 1076
8.41 Function2D.cs . . . 1085
8.42 Pie.cs . . . 1097
8.43 Plot.Area.cs . . . 1099
8.44 Plot.BarChart.cs . . . 1106
8.45 Plot.BoxPlot.cs . . . 1121
8.46 Plot.cs . . . 1129
8.47 Plot.Function1D.cs . . . 1133
8.48 Plot.Function2D.cs . . . 1135
8.49 Plot.Histogram.cs . . . 1140
8.50 Plot.Lines.cs . . . 1162
8.51 Plot.Pie.cs . . . 1170
8.52 Plot.ScatterPlot.cs . . . 1175
8.53 Plot.SwarmPlot.cs . . . 1180
8.54 Plot.ViolinPlot.cs . . . 1188
8.55 Swarm.cs . . . 1196
8.56 Trendlines.cs . . . 1198
8.57 Violin.cs . . . 1213
8.58 GradientBrushApplicator.cs . . . 1217
8.59 ImageSharpContext.cs . . . 1220
8.60 Raster.cs . . . 1240
8.61 SVGContext.cs . . . 1244
8.62 SVGParser.cs . . . 1299
8.63 Lights.cs . . . 1346
8.64 Materials.cs . . . 1357
8.65 ObjectFactory.cs . . . 1359
8.66 Scene.cs . . . 1365
8.67 Animation.cs . . . 1367
8.68 Brush.cs . . . 1388
8.69 Colour.cs . . . 1395
8.70 Document.cs . . . 1403
8.71 Enums.cs . . . 1405

8.72 BoxBlurFilter.cs . . . 1409
8.73 ColourMatrixFilter.cs . . . 1415
8.74 CompositeFilter.cs . . . 1423
8.75 ConvolutionFilter.cs . . . 1425
8.76 Filters.cs . . . 1429
8.77 GaussianBlurFilter.cs . . . 1432
8.78 MaskFilter.cs . . . 1436
8.79 Font.cs . . . 1438
8.80 FontLibrary.cs . . . 1447
8.81 FormattedText.cs . . . 1457
8.82 Gradients.cs . . . 1467
8.83 Graphics.cs . . . 1479
8.84 Graphics.Text.cs . . . 1501
8.85 GraphicsAction.cs . . . 1519
8.86 GraphicsPath.cs . . . 1523
8.87 ImageFormats.cs . . . 1580
8.88 Point.cs . . . 1594
8.89 RasterImage.cs . . . 1600
8.90 Segment.cs . . . 1604
8.91 SmoothSpline.cs . . . 1619
8.92 StandardColours.cs . . . 1620
8.93 TrueType.cs . . . 1629

Index **1691**

Chapter 1

VectSharp: a light library for C# vector graphics

Full documentation with interactive examples

1.1 Introduction

VectSharp is a library to create vector graphics (including text) in C#, without too many dependencies.

VectSharp is written using .NET Core, and is available for Mac, Windows and Linux. Since version 2.0.0, it is released under an LGPLv3 license. It includes 14 standard fonts, originally released under an ASL-2.0 license.

It includes an abstract layer on top of which output layers can be written. Currently, there are five available output layers:

- **VectSharp.PDF** produces PDF documents.
- **VectSharp.Canvas** produces an `Avalonia.Controls.Canvas` object (`https://avaloniaui.net/docs/controls/canvas`) containing the rendered graphics objects.
- **VectSharp.SVG** produces vector graphics in SVG format.
- **VectSharp.Raster** produces raster images in PNG format, (this is done by rendering the image to a PDF document, and then using the `MuPDFCore` library to render the PDF). Since version 2.0.0, VectSharp.Raster is released under an AGPLv3 licence.
- **VectSharp.Raster.ImageSharp** produces raster images in multiple formats (BMP, GIF, JPEG, PBM, PNG, TGA, TIFF, WebP) using the `SixLabors.ImageSharp` library.

VectSharp.Raster and VectSharp.Raster.ImageSharp are somewhat overlapping, as both of them can be used to create PNG images. However, VectSharp.Raster is much faster, though it only supports the PNG format. Instead, VectSharp.Raster.ImageSharp is slower, but supports more formats and has a more permissive licence. Another difference is that VectSharp.Raster carries a native dependency (through MuPDFCore), while VectSharp.ImageSharp does not.

Furthermore:

- The **`VectSharp.Plots`** package contains classes and methods to draw plots (such as scatter plots, line charts, bar charts, box plots, function plots, and more).

- **VectSharp.ThreeD** adds support for 3D vector and raster graphics.
- **VectSharp.Markdown** can be used to transform Markdown documents into VectSharp objects, that can then be exported e.g. as PDF or SVG files, or displayed in an Avalonia `Canvas`. **VectSharp.MarkdownCanvas** uses VectSharp.Markdown to render Markdown documents in Avalonia applications (an example of this is in the `MarkdownViewerDemo project`).
- **VectSharp.MuPDFUtils**, released under an AGPLv3 licence, contains some utility functions that use `MuPDFCore` to make it possible to include in VectSharp graphics images in various formats.
- **VectSharp.ImageSharpUtils** adds the same capabilities as VectSharp.MuPDFUtils, using ImageSharp instead of MuPDFCore; as a result, it is released under a more permissive LGPLv3 licence.
- **VectSharp.Fonts.Nimbus** is a package released under a GPLv3 license, which contains the standard fonts that were used in VectSharp before version 2.0.0. Since these fonts are released under a GPL license, they had to be replaced when the VectSharp license changed to LGPL. See the [Font libraries](section) below for information on how to re-enable these fonts.
- The **Animation** class (provided in the base VectSharp package) can be used to create animations that can be saved as animated GIFs (using VectSharp.Raster.ImageSharp), SVGs (using VectSharp.SVG) and PNGs (using VectSharp.Raster or VectSharp.Raster.ImageSharp).

1.2 Installing VectSharp

To include VectSharp in your project, you will need one of the output layer NuGet packages: `VectSharp.PDF`, `VectSharp.Canvas`, `VectSharp.Raster`, `VectSharp.Raster.ImageSharp`, or `VectSharp.SVG`. You will need `VectSharp.ThreeD` to work with 3D graphics or `VectSharp.Plots` to create plots. You may want the `VectSharp.MuPDFUtils` package if you wish to manipulate raster images, and the `VectSharp.Fonts.Nimbus` if you want to restore the GPL-licensed fonts used in previous versions of the library.

1.3 Usage

You can find detailed documentation for the VectSharp library, **including interactive examples**, at the `documentation website`. A comprehensive API reference is also available, both as a `website` and as a `PDF manual`.

In general, working with VectSharp involves: creating a `Document`, adding `Page`s, drawing to the `Page`s' `Graphics` objects and, finally, exporting them to a PDF document, `Canvas`, PNG image or SVG document.

- Create a `Document`:
```
using VectSharp;
// ...
Document doc = new Document();
```
- Add a `Page`:
```
doc.Pages.Add(new Page(1000, 1000));
```
- Draw to the `Page`'s `Graphics` object:
```
Graphics gpr = doc.Pages.Last().Graphics;
gpr.FillRectangle(100, 100, 800, 800, Colour.FromRgb(128, 128, 128));
```
- Save as PDF document:
```
using VectSharp.PDF;
//...
doc.SaveAsPDF(@"Sample.pdf");
```

- Export the graphics to a `Canvas`:
```
using VectSharp.Canvas;
//...
Avalonia.Controls.Canvas can = doc.Pages.Last().PaintToCanvas();
```

- Export the graphics to a `Canvas`, using a multi-layer, multi-threaded, triple-buffered renderer based on SkiaSharp (which provides the best performance if you wish e.g. to place the canvas within a `Zoom↩Border`):
```
using VectSharp.Canvas;
//...
// A single page
Avalonia.Controls.Canvas can = doc.Pages.Last().PaintToSKCanvas();
// The whole document - each page will correspond to a layer
Avalonia.Controls.Canvas can = doc.PaintToSKCanvas();
```

- Save as a PNG image:
```
using VectSharp.Raster;
//...
doc.Pages.Last().SaveAsPNG(@"Sample.png");
```

- Save as a JPEG image:
```
using VectSharp.Raster.ImageSharp;
//...
doc.Pages.Last().SaveAsImage(@"Sample.jpg");
```

- Save as an SVG document:
```
using VectSharp.SVG;
//...
doc.Pages.Last().SaveAsSVG(@"Sample.svg");
```

- PDF and SVG documents support both internal and external links:
```
using VectSharp;
using VectSharp.PDF;
using VectSharp.SVG;
//...
Document document = new Document();
Page page = new Page(1000, 1000);
document.Pages.Add(page);
page.Graphics.FillRectangle(100, 100, 800, 50, Colour.FromRgb(128, 128, 128), tag:  "linkToGitHub");
page.Graphics.FillRectangle(100, 300, 800, 50, Colour.FromRgb(255, 0, 0), tag:
"linkToBlueRectangle");
page.Graphics.FillRectangle(100, 850, 800, 50, Colour.FromRgb(0, 0, 255), tag:  "blueRectangle");
Dictionary<string, string> links = new Dictionary<string, string>() { { "linkToGitHub",
"https://github.com/" }, { "linkToBlueRectangle", "#blueRectangle" } };
page.SaveAsSVG(@"Links.svg", linkDestinations:  links);
document.SaveAsPDF(@"Links.pdf", linkDestinations:  links);
```

  This code produces a document with three rectangles: the grey one at the top links to the GitHub home page, while the red one in the middle is a hyperlink to the blue one at the bottom. Links in PDF documents can refer to objects that are in a different page than the one containing the link.

The public classes and methods are `fully documented` (with interactive examples created using Blazor), and you can find a (much) more detailed code example in `MainWindow.xaml.cs`. A detailed guide about 3D graphics in VectSharp.ThreeD is available in the `VectSharp.ThreeD` folder. Further example code for animations is available in the `DemoAnimation project`.

1.4 Font libraries

Since version 2.0.0, font names are resolved using a "font library". This is a class that implements the `VectSharp.IFontLibrary` interface, providing methods to obtain a `FontFamily` object from a `string` or a `FontFamily.StandardFontFamilies` enumeration. The default font library included in VectSharp uses the embedded fonts (Arimo, Tinos, Cousine) as the standard font families.

In practice, assuming you want to use the default font library, you have the following options to create a `Font↩Family` object:
```
using VectSharp;
// ...
FontFamily helvetica = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica); // Will
      resolve to the Arimo font family.
FontFamily times = FontFamily.ResolveFontFamily("Times-Roman"); // Will resolve to the Tinos font family.
```

These replace the `FontFamily(string)` and `FontFamily(StandardFontFamilies)` constructors of previous versions of VectSharp. Overloads of this method let you specify a list of "fallback" fonts that will be used if the first font you specify is not available.

If you wish, you can replace the default font library with a different one; this will change the way font families are resolved. For example, after installing the `VectSharp.Fonts.Nimbus` NuGet package, you can do:

```
using VectSharp;
// ...
FontFamily.DefaultFontLibrary = VectSharp.Fonts.Nimbus.Library;
FontFamily helvetica = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica); // Will
      resolve to the Nimbus Sans L font family.
FontFamily times = FontFamily.ResolveFontFamily("Times-Roman"); // Will resolve to the Nimbus Roman No 9 L
      font family.
```

This will let you re-enable the fonts that were used in previous versions of VectSharp.

You can also use multiple font libraries in the same project. Again, assuming you have installed the VectSharp.Fonts.Nimbus NuGet package:

```
using VectSharp;
FontFamily helvetica1 = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica); // Will
      resolve to the Arimo font family.
FontFamily times1 = FontFamily.ResolveFontFamily("Times-Roman"); // Will resolve to the Tinos font family.
FontFamily helvetica2 = VectSharp.Fonts.Nimbus.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica);
      // Will resolve to the Nimbus Sans L font family.
FontFamily times2 = VectSharp.Fonts.Nimbus.ResolveFontFamily("Times-Roman"); // Will resolve to the Nimbus
      Roman No 9 L font family.
```

Finally, you can create your own font library class (which could implement things such as dowloading fonts from Google Fonts, or finding them in the user's system font directory...) by creating a class that implements the `IFont↩Library` interface or that extends the `FontLibrary` class (in this latter case, you get a default implementation for the `ResolveFontFamily` overloads that use a list of fallback fonts).

1.5 Creating new output layers

VectSharp can be easily extended to provide additional output layers. To do so:

1. Create a new class implementing the `IGraphicsContext` interface.
2. Provide an extension method to either the `Page` or `Document` types.
3. Somewhere in the extension method, call the `CopyToIGraphicsContext` method on the `Graphics` object of the `Page`s.
4. Opportunely save or return the rendered result.

1.6 Compiling VectSharp from source

The VectSharp source code includes an example project (*VectSharp.Demo*) presenting how VectSharp can be used to produce graphics.

To be able to compile VectSharp from source, you will need to install the latest `.NET SDK` for your operating system.

You can use `Microsoft Visual Studio` to compile the program. The following instructions will cover compiling VectSharp from the command line, instead.

First of all, you will need to download the VectSharp source code: `VectSharp.tar.gz` and extract it somewhere.

1.6.1 Windows

Open a command-line window in the folder where you have extracted the source code, and type:

```
BuildDemo <Target>
```

Where `<Target>` can be one of `Win-x64`, `Linux-x64` or `Mac-x64` depending on which platform you wish to generate executables for.

In the Release folder and in the appropriate subfolder for the target platform you selected, you will find the compiled program.

1.6.2 macOS and Linux

Open a terminal in the folder where you have extracted the source code, and type:

```
./BuildDemo.sh <Target>
```

Where `<Target>` can be one of `Win-x64`, `Linux-x64` or `Mac-x64` depending on which platform you wish to generate executables for.

In the Release folder and in the appropriate subfolder for the target platform you selected, you will find the compiled program.

If you receive an error about permissions being denied, try typing `chmod +x BuildDemo.sh` first.

1.7 Note about VectSharp.MuPDFUtils and .NET Framework

If you wish to use VectSharp.MuPDFUtils in a .NET Framework project, you will need to manually copy the native MuPDFWrapper library for the platform you are using to the executable directory (this is done automatically if you target .NET core).

One way to obtain the appropriate library files is:

1. Manually download the NuGet package for `MuPFDCore` (click on the "Download package" link on the right).
2. Rename the `.nupkg` file so that it has a `.zip` extension.
3. Extract the zip file.
4. Within the extracted folder, the library files are in the `runtimes/xxx-yyy/native/` folder, where `xxx` is either `linux`, `osx` or `win`, depending on the platform you are using, and `yyy` is `x64`, `x86` or `arm64` depending on the architecture.

Make sure you copy the appropriate file to the same folder as the executable!

Chapter 2

Namespace Index

2.1 Package List

Here are the packages with brief descriptions (if available):

VectSharp 23
VectSharp.Canvas 27
VectSharp.Filters 28
VectSharp.Fonts 28
VectSharp.ImageSharpUtils 28
VectSharp.Markdown 29
VectSharp.MarkdownCanvas 29
VectSharp.MuPDFUtils 30
VectSharp.PDF 30
VectSharp.PDF.Figures 30
VectSharp.PDF.OptionalContentGroups 30
VectSharp.Plots 31
VectSharp.Raster 33
VectSharp.Raster.ImageSharp 33
VectSharp.SVG 34
VectSharp.ThreeD 34

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

VectSharp.AnimatedPNG . . . 44
VectSharp.Animation . . . 45
VectSharp.PDF.AnnotationStyle . . . 50
VectSharp.Canvas.AvaloniaContextInterpreter . . . 66
VectSharp.Plots.Bars<(T, double)> . . . 70
  VectSharp.Plots.CategoricalBars< T > . . . 92
VectSharp.TrueTypeFile.Bearings . . . 76
VectSharp.Brush . . . 90
  VectSharp.GradientBrush . . . 281
    VectSharp.LinearGradientBrush . . . 427
    VectSharp.RadialGradientBrush . . . 647
  VectSharp.SolidColourBrush . . . 737
VectSharp.TrueTypeFile.ClassDefinitionTable.ClassRangeRecord . . . 97
VectSharp.Filters.ColourMatrix . . . 120
VectSharp.Colours . . . 136
Avalonia.Controls.Control
  VectSharp.Canvas.AnimatedCanvas . . . 40
  VectSharp.Canvas.SKMultiLayerRenderCanvas . . . 706
VectSharp.Font.DetailedFontMetrics . . . 215
VectSharp.Document . . . 219
EventArgs
  VectSharp.Markdown.BlockRenderedEventArgs . . . 77
  VectSharp.Markdown.BlockRenderingEventArgs . . . 79
  VectSharp.Markdown.InlineRenderedEventArgs . . . 402
  VectSharp.Markdown.InlineRenderingEventArgs . . . 404
  VectSharp.Markdown.LineEventArgs . . . 439
  VectSharp.Markdown.LineMeasuredEventArgs . . . 446
  VectSharp.Markdown.PageFinishedEventArgs . . . 576
  VectSharp.Markdown.PageStartedEventArgs . . . 578
Exception
  VectSharp.FontFamilyCreationException . . . 252
  VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.UnknownFormatException . . . 803
  VectSharp.UnbalancedStackException . . . 798
VectSharp.Canvas.FilterOption . . . 225
VectSharp.PDF.PDFContextInterpreter.FilterOption . . . 228

VectSharp.SVG.SVGContextInterpreter.FilterOption . . . 231
VectSharp.Font . . . 238
VectSharp.FontFamily . . . 245
VectSharp.ResourceFontFamily . . . 686
VectSharp.Font.FontUnderline . . . 257
VectSharp.Markdown.FormattedString . . . 259
VectSharp.FormattedText . . . 261
VectSharp.FormattedTextExtensions . . . 265
VectSharp.Frame . . . 267
VectSharp.Plots.Function2DGrid . . . 272
VectSharp.Gradients . . . 282
VectSharp.GradientStop . . . 291
VectSharp.Graphics . . . 297
VectSharp.GraphicsPath . . . 337
VectSharp.Markdown.HTTPUtils . . . 360
ICoordinateSystem
VectSharp.Plots.ICoordinateSystem< T > . . . 366
VectSharp.Plots.CoordinateSystem< T > . . . 201
VectSharp.Plots.ICoordinateSystem1D< T > . . . 367
VectSharp.Plots.CategoricalCoordinateSystem1D< T > . . . 94
VectSharp.Plots.CoordinateSystem1D< T > . . . 203
VectSharp.Plots.ICoordinateSystem1D< double > . . . 367
VectSharp.Plots.LinearCoordinateSystem1D . . . 415
VectSharp.Plots.LogarithmicCoordinateSystem1D . . . 457
VectSharp.Plots.ICoordinateSystem< IReadOnlyList< double > > . . . 366
VectSharp.Plots.IContinuousCoordinateSystem . . . 362
VectSharp.Plots.IContinuousInvertibleCoordinateSystem . . . 365
VectSharp.Plots.LinLogCoordinateSystem2D . . . 449
VectSharp.Plots.LinearCoordinateSystem2D . . . 418
VectSharp.Plots.LogLinCoordinateSystem2D . . . 472
VectSharp.Plots.LogarithmicCoordinateSystem2D . . . 459
VectSharp.Plots.ICoordinateSystem<(T1, T2)> . . . 366
VectSharp.Plots.CompositeCoordinateSystem2D< T1, T2 > . . . 172
VectSharp.Plots.IDataPointElement . . . 368
VectSharp.Plots.ActionDataPointElement . . . 35
VectSharp.Plots.GraphicsDataPointElement . . . 335
VectSharp.Plots.PathDataPointElement . . . 588
IDictionary
VectSharp.PDF.AnnotationStyleCollection . . . 52
IDisposable
VectSharp.AnimatedPNG.CompressedFrame . . . 177
VectSharp.Canvas.SKMultiLayerRenderCanvas . . . 706
VectSharp.Canvas.SKRenderAction . . . 715
VectSharp.DisposableIntPtr . . . 217
VectSharp.Filters.FilterWithRasterisableParameter . . . 234
VectSharp.Filters.MaskFilter . . . 522
VectSharp.RasterImage . . . 656
VectSharp.ImageSharpUtils.RasterImageFile . . . 661
VectSharp.ImageSharpUtils.RasterImageStream . . . 663
VectSharp.MuPDFUtils.RasterImageFile . . . 662
VectSharp.MuPDFUtils.RasterImageStream . . . 665
VectSharp.IEasing . . . 370
VectSharp.LinearEasing . . . 426
VectSharp.SplineEasing . . . 740
IEquatable
VectSharp.Colour . . . 104

VectSharp.TextSpacing . . . 775
VectSharp.Filters.IFilter . . . 371
VectSharp.Filters.IFilterWithLocation . . . 373
VectSharp.Filters.MaskFilter . . . 522
VectSharp.Filters.ILocationInvariantFilter . . . 390
VectSharp.Filters.BoxBlurFilter . . . 81
VectSharp.Filters.ColourMatrixFilter . . . 133
VectSharp.Filters.CompositeLocationInvariantFilter . . . 174
VectSharp.Filters.ConvolutionFilter . . . 197
VectSharp.Filters.GaussianBlurFilter . . . 278
VectSharp.Filters.IFilterWithRasterisableParameter . . . 374
VectSharp.Filters.FilterWithRasterisableParameter . . . 234
VectSharp.IFontLibrary . . . 375
VectSharp.FontLibrary . . . 254
VectSharp.DefaultFontLibrary . . . 213
VectSharp.FolderFontLibrary . . . 235
VectSharp.MultiFontLibrary . . . 529
VectSharp.SimpleFontLibrary . . . 699
VectSharp.IGraphicsContext . . . 378
VectSharp.ThreeD.ILightSource . . . 388
VectSharp.ThreeD.AmbientLightSource . . . 37
VectSharp.ThreeD.AreaLightSource . . . 61
VectSharp.ThreeD.MaskedLightSource . . . 517
VectSharp.ThreeD.ParallelLightSource . . . 581
VectSharp.ThreeD.PointLightSource . . . 633
VectSharp.ThreeD.SpotlightLightSource . . . 743
IList
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeNode . . . 558
VectSharp.PDF.OutlineTree . . . 563
VectSharp.PDF.OutlineTreeNode . . . 568
VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter . . . 391
VectSharp.ImageSharpUtils.ImageURIParser . . . 398
VectSharp.MuPDFUtils.ImageURIParser . . . 399
VectSharp.ThreeD.IMaterial . . . 400
VectSharp.ThreeD.ColourMaterial . . . 118
VectSharp.ThreeD.PhongMaterial . . . 606
VectSharp.Plots.IPlotElement . . . 406
VectSharp.Plots.Area< T > . . . 56
VectSharp.Plots.Bars< T > . . . 70
VectSharp.Plots.BoxPlot . . . 84
VectSharp.Plots.ClusteredBars . . . 99
VectSharp.Plots.ContinuousAxis . . . 180
VectSharp.Plots.ContinuousAxisLabels . . . 183
VectSharp.Plots.ContinuousAxisTicks . . . 188
VectSharp.Plots.ContinuousAxisTitle . . . 192
VectSharp.Plots.DataLabels< T > . . . 206
VectSharp.Plots.DataLine< T > . . . 210
VectSharp.Plots.ExponentialTrendLine . . . 220
VectSharp.Plots.Function2D . . . 268
VectSharp.Plots.Grid . . . 357
VectSharp.Plots.LinearTrendLine . . . 431
VectSharp.Plots.LogarithmicTrendLine . . . 467
VectSharp.Plots.MovingAverageTrendLine . . . 525
VectSharp.Plots.Pie . . . 609
VectSharp.Plots.PlotElement< T > . . . 619
VectSharp.Plots.PolynomialTrendLine . . . 637

VectSharp.Plots.PowerLawTrendLine . . . 642
VectSharp.Plots.ScatterPoints< T > . . . 687
VectSharp.Plots.StackedBars . . . 747
VectSharp.Plots.Swarm . . . 761
VectSharp.Plots.TextLabel< T > . . . 771
VectSharp.Plots.Violin . . . 805
IReadOnlyList
VectSharp.GradientStops . . . 293
VectSharp.Point . . . 624
VectSharp.ThreeD.IScene . . . 409
VectSharp.ThreeD.Scene . . . 691
VectSharp.ThreeD.LightIntensity . . . 412
VectSharp.Markdown.Line . . . 414
VectSharp.LineDash . . . 436
VectSharp.Markdown.LineFragment . . . 443
VectSharp.Markdown.RectangleFragment . . . 674
VectSharp.Markdown.TextFragment . . . 766
VectSharp.Markdown.UnderlineFragment . . . 798
VectSharp.Markdown.Margins . . . 479
VectSharp.Markdown.MarkdownContext . . . 486
VectSharp.Markdown.MarkdownEmojiExtension . . . 493
VectSharp.Markdown.MarkdownRenderer . . . 494
VectSharp.Fonts.Nimbus . . . 532
VectSharp.ThreeD.ObjectFactory . . . 533
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupExpression . . . 546
VectSharp.PDF.OptionalContentGroups.OptionalContentGroup . . . 540
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupAndExpression . . . 544
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupNotExpression . . . 550
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupOrExpression . . . 551
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupSettings . . . 553
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeLabel . . . 555
VectSharp.Page . . . 573
VectSharp.TrueTypeFile.PairKerning . . . 580
VectSharp.SVG.Parser . . . 585
VectSharp.PDF.PDFContextInterpreter . . . 590
VectSharp.PDF.PDFDocument . . . 593
VectSharp.PDF.PDFMetadata . . . 596
VectSharp.MuPDFUtils.PDFParser . . . 601
VectSharp.Plots.Plot . . . 614
VectSharp.Plots.PlotElementPresentationAttributes . . . 621
VectSharp.TrueTypeFile.CoverageTable.RangeRecord . . . 650
VectSharp.Raster.Raster . . . 651
VectSharp.Rectangle . . . 667
VectSharp.Canvas.RenderAction . . . 678
VectSharp.Segment . . . 694
VectSharp.Size . . . 705
VectSharp.Canvas.SKRenderContext . . . 727
VectSharp.Canvas.SKRenderContextInterpreter . . . 728
VectSharp.SVG.SVGContextInterpreter . . . 752
VectSharp.Markdown.SyntaxHighlighter . . . 765
VectSharp.Transition . . . 779
VectSharp.TrueTypeFile . . . 782
VectSharp.TrueTypeFile.TrueTypeName . . . 795
VectSharp.TrueTypeFile.TrueTypePoint . . . 796
UserControl
VectSharp.MarkdownCanvas.MarkdownCanvasControl . . . 481
VectSharp.TrueTypeFile.VerticalMetrics . . . 804

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

VectSharp.Plots.ActionDataPointElement
A symbol drawn by a custom Action. . . . 35
VectSharp.ThreeD.AmbientLightSource
Represents a uniform ambien light source. . . . 37
VectSharp.Canvas.AnimatedCanvas
An Avalonia.Controls.Canvas containing an animation. . . . 40
VectSharp.AnimatedPNG
Contains methods to create animated PNG image files. . . . 44
VectSharp.Animation
Describes an animation constituted by a number of frames and transitions between them. . . . 45
VectSharp.PDF.AnnotationStyle
Describes the appearance of a PDF annotation. . . . 50
VectSharp.PDF.AnnotationStyleCollection
Represents annotation style settings. . . . 52
VectSharp.Plots.Area< T >
A plot element that fills an area between a line passing through some data points and a base line. 56
VectSharp.ThreeD.AreaLightSource
Represents a light source emitting light from a circular area. . . . 61
VectSharp.Canvas.AvaloniaContextInterpreter
Contains methods to render a Page to an Avalonia.Controls.Canvas. . . . 66
VectSharp.Plots.Bars< T >
A plot element that draws bars. . . . 70
VectSharp.TrueTypeFile.Bearings
Represents the left- and right-side bearings of a glyph. . . . 76
VectSharp.Markdown.BlockRenderedEventArgs
EventArgs for the MarkdownRenderer.BlockRendered event. . . . 77
VectSharp.Markdown.BlockRenderingEventArgs
EventArgs for the MarkdownRenderer.BlockRendering event. . . . 79
VectSharp.Filters.BoxBlurFilter
Represents a filter applying a box blur. . . . 81
VectSharp.Plots.BoxPlot
A plot element that draws a box plot. . . . 84
VectSharp.Brush
Represents a brush used to fill or stroke graphics elements. This could be a solid colour, or a more complicated gradient or pattern. . . . 90

VectSharp.Plots.CategoricalBars< T >
A plot element that draws bars for categorical data. . . . 92
VectSharp.Plots.CategoricalCoordinateSystem1D< T >
Represents a categorical 1-D coordinate system. . . . 94
VectSharp.TrueTypeFile.ClassDefinitionTable.ClassRangeRecord . . . 97
VectSharp.Plots.ClusteredBars
A plot element that draws clusters of bars. . . . 99
VectSharp.Colour
Represents an RGB colour. . . . 104
VectSharp.ThreeD.ColourMaterial
Represents a material that always has the same colour, regardless of light. . . . 118
VectSharp.Filters.ColourMatrix
Represents a colour transformation matrix. . . . 120
VectSharp.Filters.ColourMatrixFilter
Represents a filter that applies a Filters.ColourMatrix to the colours of the image. . . . 133
VectSharp.Colours
Standard colours. . . . 136
VectSharp.Plots.CompositeCoordinateSystem2D< T1, T2 >
Combines two ICoordinateSystem1D<T>s to produce a ICoordinateSystem<T>. . . . 172
VectSharp.Filters.CompositeLocationInvariantFilter
Represents a filter that corresponds to applying multiple ILocationInvariantFilters one after the other. . . . 174
VectSharp.AnimatedPNG.CompressedFrame
Represents an individual frame of a PNG animation. . . . 177
VectSharp.Plots.ContinuousAxis
A plot element that draws an axis line and an arrow. . . . 180
VectSharp.Plots.ContinuousAxisLabels
A plot element that draws equally spaced labels on an axis. . . . 183
VectSharp.Plots.ContinuousAxisTicks
A plot element that draws equally spaced ticks on an axis. . . . 188
VectSharp.Plots.ContinuousAxisTitle
A plot element that draws a title for an axis. . . . 192
VectSharp.Filters.ConvolutionFilter
Represents a filter that applies a matrix convolution to the image. . . . 197
VectSharp.Plots.CoordinateSystem< T >
A coordinate system using a custom method to transform data points. . . . 201
VectSharp.Plots.CoordinateSystem1D< T >
A coordinate system using a custom method to transform data points. . . . 203
VectSharp.Plots.DataLabels< T >
A plot element that draws a text label at each data point. . . . 206
VectSharp.Plots.DataLine< T >
A plot element that draws a line passing through a set of points. . . . 210
VectSharp.DefaultFontLibrary
A default font library that resolves standard families using the embedded fonts. . . . 213
VectSharp.Font.DetailedFontMetrics
Represents detailed information about the metrics of a text string when drawn with a certain font. 215
VectSharp.DisposableIntPtr
An IDisposable wrapper around an IntPtr that frees the allocated memory when it is disposed. 217
VectSharp.Document
Represents a collection of pages. . . . 219
VectSharp.Plots.ExponentialTrendLine
A plot element that draws an exponential trendline with equation `y = b * Exp(a * x)`. . . . 220
VectSharp.Canvas.FilterOption
Determines how and whether image filters are rasterised. . . . 225
VectSharp.PDF.PDFContextInterpreter.FilterOption
Determines how and whether image filters are rasterised. . . . 228
VectSharp.SVG.SVGContextInterpreter.FilterOption
Determines how and whether image filters are rasterised. . . . 231

VectSharp.Filters.FilterWithRasterisableParameter
Represents a filter with a parameter that needs to be rasterised at the same resolution as the subject image prior to applying the filter. . . . 234
VectSharp.FolderFontLibrary
A font library that resolves fonts from a folder containing TrueType files. . . . 235
VectSharp.Font
Represents a typeface with a specific size. . . . 238
VectSharp.FontFamily
Represents a typeface. . . . 245
VectSharp.FontFamilyCreationException
An exception that occurs while creating a FontFamily. . . . 252
VectSharp.FontLibrary
Abstract class with a default implementation of font family fallbacks. . . . 254
VectSharp.Font.FontUnderline
Represents options to underline text. . . . 257
VectSharp.Markdown.FormattedString
Represents a string with associated formatting information. . . . 259
VectSharp.FormattedText
Represents a run of text that should be drawn with the same style. . . . 261
VectSharp.FormattedTextExtensions
Contains extension methods for collections of FormattedText objects. . . . 265
VectSharp.Frame
A key frame for an animation. . . . 267
VectSharp.Plots.Function2D
A plot element that plots a function of two variables. . . . 268
VectSharp.Plots.Function2DGrid
Represents a function of two variables that has been sampled in some points. . . . 272
VectSharp.Filters.GaussianBlurFilter
Represents a filter that applies a Gaussian blur effect. . . . 278
VectSharp.GradientBrush
Represents a brush painting with a gradient. . . . 281
VectSharp.Gradients
Standard gradients. . . . 282
VectSharp.GradientStop
Represents a colour stop in a gradient. . . . 291
VectSharp.GradientStops
Represents a read-only list of GradientStops. . . . 293
VectSharp.Graphics
Represents an abstract drawing surface. . . . 297
VectSharp.Plots.GraphicsDataPointElement
A symbol defined by a VectSharp.Graphics object. . . . 335
VectSharp.GraphicsPath
Represents a graphics path that can be filled or stroked. . . . 337
VectSharp.Plots.Grid
A plot element that draws a grid. . . . 357
VectSharp.Markdown.HTTPUtils
Contains utilities to resolve absolute and relative URIs. . . . 360
VectSharp.Plots.IContinuousCoordinateSystem
Represents a coordinate system performing continuous transformations. . . . 362
VectSharp.Plots.IContinuousInvertibleCoordinateSystem
Represents a coordinate system performing continuous invertible transformations. . . . 365
VectSharp.Plots.ICoordinateSystem< T >
Represents a coordinate system. . . . 366
VectSharp.Plots.ICoordinateSystem1D< T >
Represents a coordinate system tranforming data points of type *T* into `double`s. . . . 367
VectSharp.Plots.IDataPointElement
Represents a symbol that can be added to the plot at a specified position. . . . 368

VectSharp.IEasing
Describes a function used to transform the transition speed. 370
VectSharp.Filters.IFilter
Represents a filter. Do not implement this interface directly; instead, implement ILocationInvariantFilter or IFilterWithLocation. 371
VectSharp.Filters.IFilterWithLocation
Represents a filter whose results depend on the position of the subject image on the graphics surface. 373
VectSharp.Filters.IFilterWithRasterisableParameter
Represents a filter with a parameter that needs to be rasterised at the same resolution as the subject image prior to applying the filter. The FilterWithRasterisableParameter abstract class provides a default implementation of this interface. 374
VectSharp.IFontLibrary
Represents a font library with methods to create FontFamily objects from a string or from FontFamily.StandardFontFamilies. 375
VectSharp.IGraphicsContext
This interface should be implemented by classes intended to provide graphics output capability to a Graphics object. 378
VectSharp.ThreeD.ILightSource
Represents a light source. 388
VectSharp.Filters.ILocationInvariantFilter
Represents a filter that can be applied to an image regardless of its location on the graphics surface. 390
VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter
Contains methods to render a Page to an Image. 391
VectSharp.ImageSharpUtils.ImageURIParser
Provides a method to parse an image URI into a page. 398
VectSharp.MuPDFUtils.ImageURIParser
Provides a method to parse an image URI into a page. 399
VectSharp.ThreeD.IMaterial
Represents a material used to the determine the appearance of Triangle3DElement. 400
VectSharp.Markdown.InlineRenderedEventArgs
EventArgs for the MarkdownRenderer.InlineRendered event. 402
VectSharp.Markdown.InlineRenderingEventArgs
EventArgs for the MarkdownRenderer.InlineRendering event. 404
VectSharp.Plots.IPlotElement
Represents a plot element. 406
VectSharp.ThreeD.IScene
Represents a 3D scene. 409
VectSharp.ThreeD.LightIntensity
Represents the intensity of a light source at a particular point. 412
VectSharp.Markdown.Line
Represents a text line. 414
VectSharp.Plots.LinearCoordinateSystem1D
Represents a 1-D linear coordinate system. 415
VectSharp.Plots.LinearCoordinateSystem2D
Represents a linear coordinate system. 418
VectSharp.LinearEasing
Describes a linear easing (i.e., no easing). 426
VectSharp.LinearGradientBrush
Represents a brush painting with a linear gradient. 427
VectSharp.Plots.LinearTrendLine
A plot element that draws a linear trendline with equation `y = a * x + b`. 431
VectSharp.LineDash
Represents instructions on how to paint a dashed line. 436
VectSharp.Markdown.LineEventArgs
EventArgs for the MarkdownRenderer.LineRendering and MarkdownRenderer.LineRendered events. 439

VectSharp.Markdown.LineFragment
Represents one of the fragments that make up a Line. 443
VectSharp.Markdown.LineMeasuredEventArgs
EventArgs for the MarkdownRenderer.LineMeasured event. 446
VectSharp.Plots.LinLogCoordinateSystem2D
Represents a semi-logarithmic coordinate system with a logarithmic transformation on the X axis. 449
VectSharp.Plots.LogarithmicCoordinateSystem1D
Represents a 1-D logarithmic coordinate system. 457
VectSharp.Plots.LogarithmicCoordinateSystem2D
Represents a logarithmic coordinate system. 459
VectSharp.Plots.LogarithmicTrendLine
A plot element that draws a logarithmic trendline with equation `y = a * Ln(x) + b`. . . . 467
VectSharp.Plots.LogLinCoordinateSystem2D
Represents a semi-logarithmic coordinate system with a logarithmic transformation on the Y axis. 472
VectSharp.Markdown.Margins
Represents the margins of a page. 479
VectSharp.MarkdownCanvas.MarkdownCanvasControl
A control to display a Markdown document in an Avalonia application. 481
VectSharp.Markdown.MarkdownContext
Represents the current status of the Markdown rendering process. 486
VectSharp.Markdown.MarkdownEmojiExtension
Contains extension methods for the MarkdownPipelineBuilder class. 493
VectSharp.Markdown.MarkdownRenderer
Renders Markdown documents into VectSharp graphics objects. 494
VectSharp.ThreeD.MaskedLightSource
Represents a point light source with a stencil in front of it. 517
VectSharp.Filters.MaskFilter
Represents a filter that uses the luminance of an image to mask another image. 522
VectSharp.Plots.MovingAverageTrendLine
A plot element that draws a moving average trendline. 525
VectSharp.MultiFontLibrary
A font library that tries to resolve fonts using other font libraries. 529
VectSharp.Fonts.Nimbus
Contains an IFontLibrary providing access to the Nimbus family of standard fonts (used e.g. by MuPDF). 532
VectSharp.ThreeD.ObjectFactory
A static class containing methods to create complex 3D objects. 533
VectSharp.PDF.OptionalContentGroups.OptionalContentGroup
Represents an optional content group. 540
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupAndExpression
An expression that is true when all its arguments are true. 544
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupExpression
Represents the result of a boolean operation on OptionalContentGroups. 546
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupNotExpression
An expression that is true when its argument is false. 550
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupOrExpression
An expression that is true when any of its arguments are true. 551
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupSettings
Represents the settings for optional content groups (sometimes referred to as layers) in a PDF document. 553
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeLabel
A label for an OptionalContentGroupTreeNode. 555
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeNode
Represents a node in an optional content group presentation tree. 558
VectSharp.PDF.OutlineTree
Represents an outline tree (table of contents) for a PDF document. 563
VectSharp.PDF.OutlineTreeNode
An item in an outline tree. 568

VectSharp.Page
Represents a Graphics object with a width and height. . . . 573
VectSharp.Markdown.PageFinishedEventArgs
EventArgs for the MarkdownRenderer.PageFinished event. . . . 576
VectSharp.Markdown.PageStartedEventArgs
EventArgs for the MarkdownRenderer.PageStarted event. . . . 578
VectSharp.TrueTypeFile.PairKerning
Contains information describing how the position of two glyphs in a kerning pair should be altered. . . . 580
VectSharp.ThreeD.ParallelLightSource
Represents a parallel light source. . . . 581
VectSharp.SVG.Parser
Contains methods to read an SVG image file. . . . 585
VectSharp.Plots.PathDataPointElement
A symbol defined by a GraphicsPath. . . . 588
VectSharp.PDF.PDFContextInterpreter
Contains methods to render a Document as a PDF document. . . . 590
VectSharp.PDF.PDFDocument
A low-level representation of a PDF document. . . . 593
VectSharp.PDF.PDFMetadata
Represents metadata for a PDF document. . . . 596
VectSharp.MuPDFUtils.PDFParser
Contains methods to import PDF documents as vectors. . . . 601
VectSharp.ThreeD.PhongMaterial
Represents a material that uses a Phong reflection model to determine the colour of the material based on the light sources that hit it. . . . 606
VectSharp.Plots.Pie
A plot element that draws a pie or a doughnut. . . . 609
VectSharp.Plots.Plot
Represents a collection of plot elements. . . . 614
VectSharp.Plots.PlotElement< T >
A plot element that uses an Action to draw its contents. . . . 619
VectSharp.Plots.PlotElementPresentationAttributes
Determines the appearance of plot elements. . . . 621
VectSharp.Point
Represents a point relative to an origin in the top-left corner. . . . 624
VectSharp.ThreeD.PointLightSource
Represents a point light source. . . . 633
VectSharp.Plots.PolynomialTrendLine
A plot element that draws a polynomial trendline with equation `y = a0 + a1 * x + a2 * x^2 + ... + aN * x^N`. . . . 637
VectSharp.Plots.PowerLawTrendLine
A plot element that draws a power law trendline with equation `y = b * x^a`. . . . 642
VectSharp.RadialGradientBrush
Represents a brush painting with a radial gradient. . . . 647
VectSharp.TrueTypeFile.CoverageTable.RangeRecord . . . 650
VectSharp.Raster.Raster
Contains methods to render a page to a PNG image. . . . 651
VectSharp.RasterImage
Represents a raster image, created from raw pixel data. Consider using the derived classes included in the NuGet package "VectSharp.MuPDFUtils" if you need to load a raster image from a file or a Stream. . . . 656
VectSharp.ImageSharpUtils.RasterImageFile
A RasterImage created from a file. . . . 661
VectSharp.MuPDFUtils.RasterImageFile
A RasterImage created from a file. . . . 662
VectSharp.ImageSharpUtils.RasterImageStream
A RasterImage created from a stream. . . . 663

VectSharp.MuPDFUtils.RasterImageStream
A RasterImage created from a stream. . . . 665
VectSharp.Rectangle
Represents a rectangle. . . . 667
VectSharp.Markdown.RectangleFragment
Represents a rectangle. . . . 674
VectSharp.Canvas.RenderAction
Represents a light-weight rendering action. . . . 678
VectSharp.ResourceFontFamily
Represents a FontFamily created from a resource stream. . . . 686
VectSharp.Plots.ScatterPoints< T >
A plot element that draws a symbol at the location of multiple data points. . . . 687
VectSharp.ThreeD.Scene
Represents a 3D scene. . . . 691
VectSharp.Segment
Represents a segment as part of a GraphicsPath. . . . 694
VectSharp.SimpleFontLibrary
A font library that can be used to cache and resolve font family names. . . . 699
VectSharp.Size
Represents the size of an object. . . . 705
VectSharp.Canvas.SKMultiLayerRenderCanvas
Represents a multi-threaded, triple-buffered canvas on which the image is drawn using SkiaSharp. . . . 706
VectSharp.Canvas.SKRenderAction
Represents a light-weight rendering action. . . . 715
VectSharp.Canvas.SKRenderContext
Represents a page that has been prepared for fast rendering using the SkiaSharp renderer. . . . 727
VectSharp.Canvas.SKRenderContextInterpreter
Contains methods to render a Page to an Avalonia.Controls.Canvas using the SkiaSharp renderer. . . . 728
VectSharp.SolidColourBrush
Represents a brush painting with a single solid colour. . . . 737
VectSharp.SplineEasing
Describes an easing defined by a Cubic Bezier curve. . . . 740
VectSharp.ThreeD.SpotlightLightSource
Represents a conic spotlight. . . . 743
VectSharp.Plots.StackedBars
A plot element that draws stacked bars. . . . 747
VectSharp.SVG.SVGContextInterpreter
Contains methods to render a Page as an SVG file. . . . 752
VectSharp.Plots.Swarm
A plot element that draws a swarm plot. . . . 761
VectSharp.Markdown.SyntaxHighlighter
Contains methods to perform syntax highlighting. . . . 765
VectSharp.Markdown.TextFragment
A fragment containing text. . . . 766
VectSharp.Plots.TextLabel< T >
A plot element that draws a single text label. . . . 771
VectSharp.TextSpacing
Contains information about text spacing. . . . 775
VectSharp.Transition
Describes the transition between two successive Frames. . . . 779
VectSharp.TrueTypeFile
Represents a font file in TrueType format. Reference: http://stevehanov.ca/blog/?id=143, https://developer.apple.com/fonts/TrueType-Reference-Manual/, https://docs.microsoft.com/en-us/typography/opentype/spec/ 782

VectSharp.TrueTypeFile.TrueTypeName
Represets a TrueType name. 795
VectSharp.TrueTypeFile.TrueTypePoint
Represents a point in a TrueType path description. 796
VectSharp.UnbalancedStackException
The exception that is thrown when an unbalanced graphics state stack occurs. 798
VectSharp.Markdown.UnderlineFragment
Represents a text underline. 798
VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.UnknownFormatException
The exception that is raised when the output file format is not specified and the file name does not have an extension corresponding to a known file format. 803
VectSharp.TrueTypeFile.VerticalMetrics
Represents the maximum heigth above and depth below the baseline of a glyph. 804
VectSharp.Plots.Violin
A plot element that draws a violin plot. 805

Chapter 5

File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

VectSharp.Canvas/AnimatedCanvas.cs . . . 811
VectSharp.Canvas/AvaloniaContext.cs . . . 815
VectSharp.Canvas/RenderingParameters.cs . . . 850
VectSharp.Canvas/SkiaBitmap.cs . . . 852
VectSharp.Canvas/SKMultiLayerRenderCanvas.cs . . . 853
VectSharp.Canvas/SKRenderContext.cs . . . 868
VectSharp.Fonts.Nimbus/Nimbus.cs . . . 892
VectSharp.ImageSharpUtils/ImageUriParser.cs . . . 965
VectSharp.ImageSharpUtils/RasterImages.cs . . . 974
VectSharp.Markdown/HtmlTag.cs . . . 893
VectSharp.Markdown/Line.cs . . . 899
VectSharp.Markdown/MarkdownContext.cs . . . 905
VectSharp.Markdown/MarkdownRenderer.Blocks.cs . . . 909
VectSharp.Markdown/MarkdownRenderer.cs . . . 919
VectSharp.Markdown/MarkdownRenderer.Events.cs . . . 926
VectSharp.Markdown/MarkdownRenderer.Inlines.cs . . . 933
VectSharp.Markdown/MarkdownRenderer.Properties.cs . . . 943
VectSharp.Markdown/MarkdownRenderer.Tables.cs . . . 949
VectSharp.Markdown/SyntaxHighlighting.cs . . . 953
VectSharp.Markdown/UnicodeEmojiParser.cs . . . 956
VectSharp.Markdown/Word.cs . . . 958
VectSharp.MarkdownCanvas/ImageCache.cs . . . 960
VectSharp.MarkdownCanvas/MarkdownCanvas.axaml.cs . . . 961
VectSharp.MuPDFUtils/ImageURIParser.cs . . . 966
VectSharp.MuPDFUtils/PDFParser.cs . . . 969
VectSharp.MuPDFUtils/RasterImages.cs . . . 977
VectSharp.PDF/AnnotationStyles.cs . . . 979
VectSharp.PDF/Figures.cs . . . 981
VectSharp.PDF/OptionalContentGroups.cs . . . 986
VectSharp.PDF/OutlineTree.cs . . . 993
VectSharp.PDF/PDFContext.cs . . . 996
VectSharp.PDF/PDFContextInterpreter.cs . . . 1006
VectSharp.PDF/PDFDocument.cs . . . 1018
VectSharp.PDF/PDFMetadata.cs . . . 1022
VectSharp.PDF/PDFProcessFigure.cs . . . 1024

VectSharp.Plots/Axes.cs . . . 1032
VectSharp.Plots/Bars.cs . . . 1044
VectSharp.Plots/BoxPlot.cs . . . 1058
VectSharp.Plots/CoordinateSystems.cs . . . 1062
VectSharp.Plots/DataPoints.cs . . . 1076
VectSharp.Plots/Function2D.cs . . . 1085
VectSharp.Plots/Pie.cs . . . 1097
VectSharp.Plots/Plot.Area.cs . . . 1099
VectSharp.Plots/Plot.BarChart.cs . . . 1106
VectSharp.Plots/Plot.BoxPlot.cs . . . 1121
VectSharp.Plots/Plot.cs . . . 1129
VectSharp.Plots/Plot.Function1D.cs . . . 1133
VectSharp.Plots/Plot.Function2D.cs . . . 1135
VectSharp.Plots/Plot.Histogram.cs . . . 1140
VectSharp.Plots/Plot.Lines.cs . . . 1162
VectSharp.Plots/Plot.Pie.cs . . . 1170
VectSharp.Plots/Plot.ScatterPlot.cs . . . 1175
VectSharp.Plots/Plot.SwarmPlot.cs . . . 1180
VectSharp.Plots/Plot.ViolinPlot.cs . . . 1188
VectSharp.Plots/Swarm.cs . . . 1196
VectSharp.Plots/Trendlines.cs . . . 1198
VectSharp.Plots/Violin.cs . . . 1213
VectSharp.Raster.ImageSharp/GradientBrushApplicator.cs . . . 1217
VectSharp.Raster.ImageSharp/ImageSharpContext.cs . . . 1220
VectSharp.Raster/Raster.cs . . . 1240
VectSharp.SVG/SVGContext.cs . . . 1244
VectSharp.SVG/SVGParser.cs . . . 1299
VectSharp.ThreeD/Lights.cs . . . 1346
VectSharp.ThreeD/Materials.cs . . . 1357
VectSharp.ThreeD/ObjectFactory.cs . . . 1359
VectSharp.ThreeD/Scene.cs . . . 1365
VectSharp/Animation.cs . . . 1367
VectSharp/Brush.cs . . . 1388
VectSharp/Colour.cs . . . 1395
VectSharp/Document.cs . . . 1403
VectSharp/Enums.cs . . . 1405
VectSharp/Font.cs . . . 1438
VectSharp/FontLibrary.cs . . . 1447
VectSharp/FormattedText.cs . . . 1457
VectSharp/Gradients.cs . . . 1467
VectSharp/Graphics.cs . . . 1479
VectSharp/Graphics.Text.cs . . . 1501
VectSharp/GraphicsAction.cs . . . 1519
VectSharp/GraphicsPath.cs . . . 1523
VectSharp/ImageFormats.cs . . . 1580
VectSharp/Point.cs . . . 1594
VectSharp/RasterImage.cs . . . 1600
VectSharp/Segment.cs . . . 1604
VectSharp/SmoothSpline.cs . . . 1619
VectSharp/StandardColours.cs . . . 1620
VectSharp/TrueType.cs . . . 1629
VectSharp/Filters/BoxBlurFilter.cs . . . 1409
VectSharp/Filters/ColourMatrixFilter.cs . . . 1415
VectSharp/Filters/CompositeFilter.cs . . . 1423
VectSharp/Filters/ConvolutionFilter.cs . . . 1425
VectSharp/Filters/Filters.cs . . . 1429
VectSharp/Filters/GaussianBlurFilter.cs . . . 1432
VectSharp/Filters/MaskFilter.cs . . . 1436

Chapter 6

Namespace Documentation

6.1 VectSharp Namespace Reference

Classes

- class AnimatedPNG

  Contains methods to create animated PNG image files.
- class Animation

  Describes an animation constituted by a number of frames and transitions between them.
- class Brush

  Represents a brush used to fill or stroke graphics elements. This could be a solid colour, or a more complicated gradient or pattern.
- struct Colour

  Represents an RGB colour.
- class Colours

  Standard colours.
- class DefaultFontLibrary

  A default font library that resolves standard families using the embedded fonts.
- class DisposableIntPtr

  An IDisposable wrapper around an IntPtr that frees the allocated memory when it is disposed.
- class Document

  Represents a collection of pages.
- class FolderFontLibrary

  A font library that resolves fonts from a folder containing TrueType files.
- class Font

  Represents a typeface with a specific size.
- class FontFamily

  Represents a typeface.
- class FontFamilyCreationException

  An exception that occurs while creating a FontFamily.
- class FontLibrary

  Abstract class with a default implementation of font family fallbacks.
- class FormattedText

  Represents a run of text that should be drawn with the same style.
- class FormattedTextExtensions

  Contains extension methods for collections of FormattedText objects.

- class Frame

  A key frame for an animation.
- class GradientBrush

  Represents a brush painting with a gradient.
- class Gradients

  Standard gradients.
- struct GradientStop

  Represents a colour stop in a gradient.
- class GradientStops

  Represents a read-only list of GradientStops.
- class Graphics

  Represents an abstract drawing surface.
- class GraphicsPath

  Represents a graphics path that can be filled or stroked.
- interface IEasing

  Describes a function used to transform the transition speed.
- interface IFontLibrary

  Represents a font library with methods to create FontFamily objects from a string or from FontFamily.StandardFontFamilies.
- interface IGraphicsContext

  This interface should be implemented by classes intended to provide graphics output capability to a Graphics object.
- class LinearEasing

  Describes a linear easing (i.e., no easing).
- class LinearGradientBrush

  Represents a brush painting with a linear gradient.
- struct LineDash

  Represents instructions on how to paint a dashed line.
- class MultiFontLibrary

  A font library that tries to resolve fonts using other font libraries.
- class Page

  Represents a Graphics object with a width and height.
- struct Point

  Represents a point relative to an origin in the top-left corner.
- class RadialGradientBrush

  Represents a brush painting with a radial gradient.
- class RasterImage

  Represents a raster image, created from raw pixel data. Consider using the derived classes included in the NuGet package "VectSharp.MuPDFUtils" if you need to load a raster image from a file or a Stream.
- struct Rectangle

  Represents a rectangle.
- class ResourceFontFamily

  Represents a FontFamily created from a resource stream.
- class Segment

  Represents a segment as part of a GraphicsPath.
- class SimpleFontLibrary

  A font library that can be used to cache and resolve font family names.
- struct Size

  Represents the size of an object.
- class SolidColourBrush

  Represents a brush painting with a single solid colour.
- class SplineEasing

  Describes an easing defined by a Cubic Bezier curve.

- struct TextSpacing

  Contains information about text spacing.
- class Transition

  Describes the transition between two successive Frames.
- class TrueTypeFile

  Represents a font file in TrueType format. Reference: http://stevehanov.ca/blog/?id=143, https://developer.apple.com/fonts/TrueType-Reference-Manual/, https://docs.microsoft.com/en-us/typography/opentype/spec/
- class UnbalancedStackException

  The exception that is thrown when an unbalanced graphics state stack occurs.

Enumerations

- enum TextBaselines

  Represent text baselines.
- enum TextAnchors

  Represents text anchors.
- enum LineCaps

  Represents line caps.
- enum LineJoins

  Represents line joining options.
- enum SegmentType

  Types of Segment.
- enum UnbalancedStackActions

  Represents ways to deal with unbalanced graphics state stacks.
- enum Script

  Represents the position of the text.
- enum FillRule

  Represents a rule used to determine whether a point is inside or outside of a shape.
- enum PixelFormats

  Represents the pixel format of a raster image.

6.1.1 Enumeration Type Documentation

6.1.1.1 FillRule

`enum VectSharp.FillRule`

Represents a rule used to determine whether a point is inside or outside of a shape.

Definition at line 263 of file Graphics.cs.

6.1.1.2 LineCaps

enum VectSharp.LineCaps

Represents line caps.

Definition at line 189 of file Enums.cs.

6.1.1.3 LineJoins

enum VectSharp.LineJoins

Represents line joining options.

Definition at line 210 of file Enums.cs.

6.1.1.4 PixelFormats

enum VectSharp.PixelFormats

Represents the pixel format of a raster image.

Definition at line 28 of file RasterImage.cs.

6.1.1.5 Script

enum VectSharp.Script

Represents the position of the text.

Definition at line 29 of file FormattedText.cs.

6.1.1.6 SegmentType

enum VectSharp.SegmentType

Types of Segment.

Definition at line 333 of file Enums.cs.

6.1.1.7 TextAnchors

enum VectSharp.TextAnchors

Represents text anchors.

Definition at line 53 of file Enums.cs.

6.1.1.8 TextBaselines

enum VectSharp.TextBaselines

Represent text baselines.

Definition at line 27 of file Enums.cs.

6.1.1.9 UnbalancedStackActions

enum VectSharp.UnbalancedStackActions

Represents ways to deal with unbalanced graphics state stacks.

Definition at line 364 of file Enums.cs.

6.2 VectSharp.Canvas Namespace Reference

Classes

- class AnimatedCanvas

  An Avalonia.Controls.Canvas containing an animation.
- class AvaloniaContextInterpreter

  Contains methods to render a Page to an Avalonia.Controls.Canvas.
- class FilterOption

  Determines how and whether image filters are rasterised.
- class RenderAction

  Represents a light-weight rendering action.
- class SKMultiLayerRenderCanvas

  Represents a multi-threaded, triple-buffered canvas on which the image is drawn using SkiaSharp.
- class SKRenderAction

  Represents a light-weight rendering action.
- class SKRenderContext

  Represents a page that has been prepared for fast rendering using the SkiaSharp renderer.
- class SKRenderContextInterpreter

  Contains methods to render a Page to an Avalonia.Controls.Canvas using the SkiaSharp renderer.

6.3 VectSharp.Filters Namespace Reference

Classes

- class BoxBlurFilter

  Represents a filter applying a box blur.
- class ColourMatrix

  Represents a colour transformation matrix.
- class ColourMatrixFilter

  Represents a filter that applies a Filters.ColourMatrix to the colours of the image.
- class CompositeLocationInvariantFilter

  Represents a filter that corresponds to applying multiple ILocationInvariantFilters one after the other.
- class ConvolutionFilter

  Represents a filter that applies a matrix convolution to the image.
- class FilterWithRasterisableParameter

  Represents a filter with a parameter that needs to be rasterised at the same resolution as the subject image prior to applying the filter.
- class GaussianBlurFilter

  Represents a filter that applies a Gaussian blur effect.
- interface IFilter

  Represents a filter. Do not implement this interface directly; instead, implement ILocationInvariantFilter or IFilterWithLocation.
- interface IFilterWithLocation

  Represents a filter whose results depend on the position of the subject image on the graphics surface.
- interface IFilterWithRasterisableParameter

  Represents a filter with a parameter that needs to be rasterised at the same resolution as the subject image prior to applying the filter. The FilterWithRasterisableParameter abstract class provides a default implementation of this interface.
- interface ILocationInvariantFilter

  Represents a filter that can be applied to an image regardless of its location on the graphics surface.
- class MaskFilter

  Represents a filter that uses the luminance of an image to mask another image.

6.4 VectSharp.Fonts Namespace Reference

Classes

- class Nimbus

  Contains an IFontLibrary providing access to the Nimbus family of standard fonts (used e.g. by MuPDF).

6.5 VectSharp.ImageSharpUtils Namespace Reference

Classes

- class ImageURIParser

  Provides a method to parse an image URI into a page.
- class RasterImageFile

  A RasterImage created from a file.
- class RasterImageStream

  A RasterImage created from a stream.

6.6 VectSharp.Markdown Namespace Reference

Classes

- class BlockRenderedEventArgs

  EventArgs for the MarkdownRenderer.BlockRendered event.
- class BlockRenderingEventArgs

  EventArgs for the MarkdownRenderer.BlockRendering event.
- struct FormattedString

  Represents a string with associated formatting information.
- class HTTPUtils

  Contains utilities to resolve absolute and relative URIs.
- class InlineRenderedEventArgs

  EventArgs for the MarkdownRenderer.InlineRendered event.
- class InlineRenderingEventArgs

  EventArgs for the MarkdownRenderer.InlineRendering event.
- class Line

  Represents a text line.
- class LineEventArgs

  EventArgs for the MarkdownRenderer.LineRendering and MarkdownRenderer.LineRendered events.
- class LineFragment

  Represents one of the fragments that make up a Line.
- class LineMeasuredEventArgs

  EventArgs for the MarkdownRenderer.LineMeasured event.
- class Margins

  Represents the margins of a page.
- class MarkdownContext

  Represents the current status of the Markdown rendering process.
- class MarkdownEmojiExtension

  Contains extension methods for the MarkdownPipelineBuilder class.
- class MarkdownRenderer

  Renders Markdown documents into VectSharp graphics objects.
- class PageFinishedEventArgs

  EventArgs for the MarkdownRenderer.PageFinished event.
- class PageStartedEventArgs

  EventArgs for the MarkdownRenderer.PageStarted event.
- class RectangleFragment

  Represents a rectangle.
- class SyntaxHighlighter

  Contains methods to perform syntax highlighting.
- class TextFragment

  A fragment containing text.
- class UnderlineFragment

  Represents a text underline.

6.7 VectSharp.MarkdownCanvas Namespace Reference

Classes

- class MarkdownCanvasControl

  A control to display a Markdown document in an Avalonia application.

6.8 VectSharp.MuPDFUtils Namespace Reference

Classes

- class ImageURIParser

  Provides a method to parse an image URI into a page.
- class PDFParser

  Contains methods to import PDF documents as vectors.
- class RasterImageFile

  A RasterImage created from a file.
- class RasterImageStream

  A RasterImage created from a stream.

6.9 VectSharp.PDF Namespace Reference

Classes

- class AnnotationStyle

  Describes the appearance of a PDF annotation.
- class AnnotationStyleCollection

  Represents annotation style settings.
- class OutlineTree

  Represents an outline tree (table of contents) for a PDF document.
- class OutlineTreeNode

  An item in an outline tree.
- class PDFContextInterpreter

  Contains methods to render a Document as a PDF document.
- class PDFDocument

  A low-level representation of a PDF document.
- class PDFMetadata

  Represents metadata for a PDF document.

6.10 VectSharp.PDF.Figures Namespace Reference

6.11 VectSharp.PDF.OptionalContentGroups Namespace Reference

Classes

- class OptionalContentGroup

  Represents an optional content group.
- class OptionalContentGroupAndExpression

  An expression that is true when all its arguments are true.
- class OptionalContentGroupExpression

  Represents the result of a boolean operation on OptionalContentGroups.
- class OptionalContentGroupNotExpression

  An expression that is true when its argument is false.

- class OptionalContentGroupOrExpression

  An expression that is true when any of its arguments are true.
- class OptionalContentGroupSettings

  Represents the settings for optional content groups (sometimes referred to as layers) in a PDF document.
- class OptionalContentGroupTreeLabel

  A label for an OptionalContentGroupTreeNode.
- class OptionalContentGroupTreeNode

  Represents a node in an optional content group presentation tree.

Enumerations

- enum OptionalContentGroupBooleanOperation

  Boolean operators for OptionalContentGroups.

6.11.1 Enumeration Type Documentation

6.11.1.1 OptionalContentGroupBooleanOperation

`enum VectSharp.PDF.OptionalContentGroups.OptionalContentGroupBooleanOperation`

Boolean operators for OptionalContentGroups.

Definition at line 29 of file OptionalContentGroups.cs.

6.12 VectSharp.Plots Namespace Reference

Classes

- class ActionDataPointElement

  A symbol drawn by a custom Action.
- class Area

  A plot element that fills an area between a line passing through some data points and a base line.
- class Bars

  A plot element that draws bars.
- class BoxPlot

  A plot element that draws a box plot.
- class CategoricalBars

  A plot element that draws bars for categorical data.
- class CategoricalCoordinateSystem1D

  Represents a categorical 1-D coordinate system.
- class ClusteredBars

  A plot element that draws clusters of bars.
- class CompositeCoordinateSystem2D

  Combines two ICoordinateSystem1D<T>s to produce a ICoordinateSystem<T>.
- class ContinuousAxis

A plot element that draws an axis line and an arrow.

- class ContinuousAxisLabels

  A plot element that draws equally spaced labels on an axis.
- class ContinuousAxisTicks

  A plot element that draws equally spaced ticks on an axis.
- class ContinuousAxisTitle

  A plot element that draws a title for an axis.
- class CoordinateSystem

  A coordinate system using a custom method to transform data points.
- class CoordinateSystem1D

  A coordinate system using a custom method to transform data points.
- class DataLabels

  A plot element that draws a text label at each data point.
- class DataLine

  A plot element that draws a line passing through a set of points.
- class ExponentialTrendLine

  A plot element that draws an exponential trendline with equation `y = b * Exp(a * x)`.
- class Function2D

  A plot element that plots a function of two variables.
- class Function2DGrid

  Represents a function of two variables that has been sampled in some points.
- class GraphicsDataPointElement

  A symbol defined by a VectSharp.Graphics object.
- class Grid

  A plot element that draws a grid.
- interface IContinuousCoordinateSystem

  Represents a coordinate system performing continuous transformations.
- interface IContinuousInvertibleCoordinateSystem

  Represents a coordinate system performing continuous invertible transformations.
- interface ICoordinateSystem

  Represents a coordinate system.
- interface ICoordinateSystem1D

  Represents a coordinate system tranforming data points of type T into `double`*s.*
- interface IDataPointElement

  Represents a symbol that can be added to the plot at a specified position.
- interface IPlotElement

  Represents a plot element.
- class LinearCoordinateSystem1D

  Represents a 1-D linear coordinate system.
- class LinearCoordinateSystem2D

  Represents a linear coordinate system.
- class LinearTrendLine

  A plot element that draws a linear trendline with equation `y = a * x + b`.
- class LinLogCoordinateSystem2D

  Represents a semi-logarithmic coordinate system with a logarithmic transformation on the X axis.
- class LogarithmicCoordinateSystem1D

  Represents a 1-D logarithmic coordinate system.
- class LogarithmicCoordinateSystem2D

  Represents a logarithmic coordinate system.
- class LogarithmicTrendLine

  A plot element that draws a logarithmic trendline with equation `y = a * Ln(x) + b`.

- class LogLinCoordinateSystem2D

  Represents a semi-logarithmic coordinate system with a logarithmic transformation on the Y axis.
- class MovingAverageTrendLine

  A plot element that draws a moving average trendline.
- class PathDataPointElement

  A symbol defined by a GraphicsPath.
- class Pie

  A plot element that draws a pie or a doughnut.
- class Plot

  Represents a collection of plot elements.
- class PlotElement

  A plot element that uses an Action to draw its contents.
- class PlotElementPresentationAttributes

  Determines the appearance of plot elements.
- class PolynomialTrendLine

  A plot element that draws a polynomial trendline with equation `y = a0 + a1 * x + a2 * x^2 + ... + aN * x^N`.
- class PowerLawTrendLine

  A plot element that draws a power law trendline with equation `y = b * x^a`.
- class ScatterPoints

  A plot element that draws a symbol at the location of multiple data points.
- class StackedBars

  A plot element that draws stacked bars.
- class Swarm

  A plot element that draws a swarm plot.
- class TextLabel

  A plot element that draws a single text label.
- class Violin

  A plot element that draws a violin plot.

6.13 VectSharp.Raster Namespace Reference

Classes

- class Raster

  Contains methods to render a page to a PNG image.

6.14 VectSharp.Raster.ImageSharp Namespace Reference

Classes

- class ImageSharpContextInterpreter

  Contains methods to render a Page to an Image.

Enumerations

- enum OutputFormats

  Enumeration containing the supported output formats.

6.14.1 Enumeration Type Documentation

6.14.1.1 OutputFormats

`enum VectSharp.Raster.ImageSharp.OutputFormats`

Enumeration containing the supported output formats.

Definition at line 1001 of file ImageSharpContext.cs.

6.15 VectSharp.SVG Namespace Reference

Classes

- class Parser

  Contains methods to read an SVG image file.
- class SVGContextInterpreter

  Contains methods to render a Page as an SVG file.

6.16 VectSharp.ThreeD Namespace Reference

Classes

- class AmbientLightSource

  Represents a uniform ambien light source.
- class AreaLightSource

  Represents a light source emitting light from a circular area.
- class ColourMaterial

  Represents a material that always has the same colour, regardless of light.
- interface ILightSource

  Represents a light source.
- interface IMaterial

  Represents a material used to the determine the appearance of Triangle3DElement.
- interface IScene

  Represents a 3D scene.
- struct LightIntensity

  Represents the intensity of a light source at a particular point.
- class MaskedLightSource

  Represents a point light source with a stencil in front of it.
- class ObjectFactory

  A static class containing methods to create complex 3D objects.
- class ParallelLightSource

  Represents a parallel light source.
- class PhongMaterial

  Represents a material that uses a Phong reflection model to determine the colour of the material based on the light sources that hit it.
- class PointLightSource

  Represents a point light source.
- class Scene

  Represents a 3D scene.
- class SpotlightLightSource

  Represents a conic spotlight.

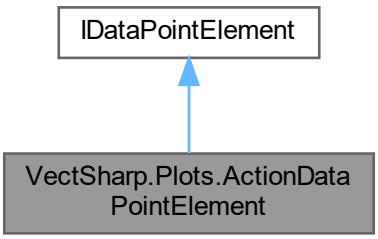
Chapter 7

Class Documentation

7.1 VectSharp.Plots.ActionDataPointElement Class Reference

A symbol drawn by a custom Action.

Inheritance diagram for VectSharp.Plots.ActionDataPointElement:



Public Member Functions

- void Plot (Graphics target, PlotElementPresentationAttributes presentationAttributes, string tag)

  Draw the symbol on the plot.

Parameters

target	*The Graphics object on which to draw. It is assumed that it has been transformed so that the symbol can be drawn centred at (0, 0)*
presentationAttributes	*Presentation attributes determining the appearance of the symbol.*
tag	*A tag to identify the symbol in the plot.*

- ActionDataPointElement (Action< Graphics, PlotElementPresentationAttributes, string > plotAction)

  Creates a new ActionDataPointElement using the specified action to draw the symbol.

Properties

- Action< Graphics, PlotElementPresentationAttributes, string > PlotAction `[get, set]`

  The Action used to draw the symbol. This should take as arguments the Graphics object on which to draw the symbol, the PlotElementPresentationAttributes describing the appearance of the symbol, and a `string` representing a tag for the symbol.

7.1.1 Detailed Description

A symbol drawn by a custom Action.

Definition at line 122 of file DataPoints.cs.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 ActionDataPointElement()

```
VectSharp.Plots.ActionDataPointElement.ActionDataPointElement (
            Action< Graphics, PlotElementPresentationAttributes, string > plotAction )
```

Creates a new ActionDataPointElement using the specified action to draw the symbol.

Parameters

plotAction	The Action used to draw the symbol. This should take as arguments the Graphics object on which to draw the symbol, the PlotElementPresentationAttributes describing the appearance of the symbol, and a `string` representing a tag for the symbol.

Definition at line 140 of file DataPoints.cs.

7.1.3 Member Function Documentation

7.1.3.1 Plot()

```
void VectSharp.Plots.ActionDataPointElement.Plot (
            Graphics target,
            PlotElementPresentationAttributes presentationAttributes,
            string tag )
```

Draw the symbol on the plot.

Parameters

target	The Graphics object on which to draw. It is assumed that it has been transformed so that the symbol can be drawn centred at (0, 0)
presentationAttributes	Presentation attributes determining the appearance of the symbol.
tag	A tag to identify the symbol in the plot.

Implements VectSharp.Plots.IDataPointElement.

7.1.4 Property Documentation

7.1.4.1 PlotAction

```
Action<Graphics, PlotElementPresentationAttributes, string> VectSharp.Plots.ActionDataPoint↩
Element.PlotAction [get], [set]
```

The Action used to draw the symbol. This should take as arguments the Graphics object on which to draw the symbol, the PlotElementPresentationAttributes describing the appearance of the symbol, and a `string` representing a tag for the symbol.

Definition at line 129 of file DataPoints.cs.

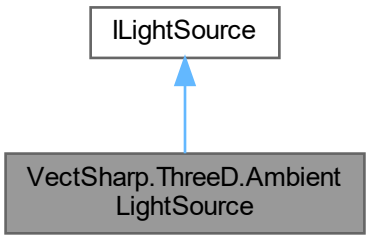
The documentation for this class was generated from the following file:

- VectSharp.Plots/DataPoints.cs

7.2 VectSharp.ThreeD.AmbientLightSource Class Reference

Represents a uniform ambien light source.

Inheritance diagram for VectSharp.ThreeD.AmbientLightSource:



Public Member Functions

- AmbientLightSource (double intensity)

  Creates a new AmbientLightSource instance.
- LightIntensity GetLightAt (Point3D point)

  Computes the light intensity at the specified point, without taking into account any obstructions.

Parameters

point	*The Point3DElement at which the light intensity should be computed.*

Returns

- double GetObstruction (Point3D point, IEnumerable< Triangle3DElement > shadowingTriangles)

  Determines the amount of obstruction of the light that results at point due to the specified shadowingTriangles .

Parameters

point	*The Point3D at which the obstruction should be computed.*
shadowingTriangles	*A collection of Triangle3DElement casting shadows.*

Returns

1 if the light is completely obstructed, 0 if the light is completely visible, a value between these if the light is partially obstructed.

Properties

- double Intensity `[get, set]`

  The intensity of the light.
- bool CastsShadow `[get]`

  Determines whether the light casts a shadow or not.

7.2.1 Detailed Description

Represents a uniform ambien light source.

Definition at line 91 of file Lights.cs.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 AmbientLightSource()

```
VectSharp.ThreeD.AmbientLightSource.AmbientLightSource (
            double intensity )
```

Creates a new AmbientLightSource instance.

Parameters

intensity	The intensity of the light.

Definition at line 105 of file Lights.cs.

7.2.3 Member Function Documentation

7.2.3.1 GetLightAt()

```
LightIntensity VectSharp.ThreeD.AmbientLightSource.GetLightAt (
            Point3D point )
```

Computes the light intensity at the specified point, without taking into account any obstructions.

Parameters

point	The Point3DElement at which the light intensity should be computed.

Returns

Implements VectSharp.ThreeD.ILightSource.

Definition at line 111 of file Lights.cs.

7.2.3.2 GetObstruction()

```
double VectSharp.ThreeD.AmbientLightSource.GetObstruction (
            Point3D point,
            IEnumerable< Triangle3DElement > shadowingTriangles )
```

Determines the amount of obstruction of the light that results at *point* due to the specified *shadowingTriangles* .

Parameters

point	The Point3D at which the obstruction should be computed.
shadowingTriangles	A collection of Triangle3DElement casting shadows.

Returns

1 if the light is completely obstructed, 0 if the light is completely visible, a value between these if the light is partially obstructed.

Implements VectSharp.ThreeD.ILightSource.

Definition at line 117 of file Lights.cs.

7.2.4 Property Documentation

7.2.4.1 CastsShadow

```
bool VectSharp.ThreeD.AmbientLightSource.CastsShadow  [get]
```

Determines whether the light casts a shadow or not.

Implements VectSharp.ThreeD.ILightSource.

Definition at line 99 of file Lights.cs.

7.2.4.2 Intensity

```
double VectSharp.ThreeD.AmbientLightSource.Intensity  [get], [set]
```

The intensity of the light.

Definition at line 96 of file Lights.cs.

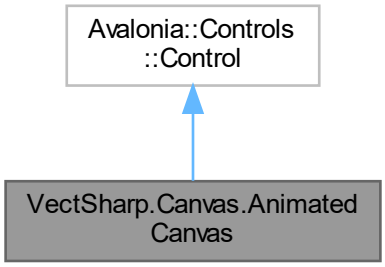
The documentation for this class was generated from the following file:

- VectSharp.ThreeD/Lights.cs

7.3 VectSharp.Canvas.AnimatedCanvas Class Reference

An Avalonia.Controls.Canvas containing an animation.

Inheritance diagram for VectSharp.Canvas.AnimatedCanvas:



Public Member Functions

- override void Render (DrawingContext context)

Static Public Attributes

- static readonly StyledProperty< int > CurrentFrameProperty

  Defines the CurrentFrame property.
- static readonly DirectProperty< AnimatedCanvas, int > FrameCountProperty = AvaloniaProperty.Register↩ Direct<AnimatedCanvas, int>(nameof(FrameCount), o => o.maxFrameIndex)

  Defines the FrameCount property.
- static readonly DirectProperty< AnimatedCanvas, double > FrameRateProperty = AvaloniaProperty.↩ RegisterDirect<AnimatedCanvas, double>(nameof(FrameRate), o => o.FrameRate)

  Defines the FrameRate property.
- static readonly StyledProperty< bool > IsPlayingProperty

  Defines the IsPlaying property.

Properties

- int CurrentFrame `[get, set]`

  The current frame in the animation.
- int FrameCount `[get]`

  The number of frames in the animation.
- double FrameRate `[get]`

  The target frame rate of the animation.
- bool IsPlaying `[get, set]`

  The current frame in the animation.

7.3.1 Detailed Description

An Avalonia.Controls.Canvas containing an animation.

Definition at line 28 of file AnimatedCanvas.cs.

7.3.2 Member Function Documentation

7.3.2.1 Render()

```
override void VectSharp.Canvas.AnimatedCanvas.Render (
            DrawingContext context )
```

Definition at line 203 of file AnimatedCanvas.cs.

7.3.3 Member Data Documentation

7.3.3.1 CurrentFrameProperty

```
readonly StyledProperty<int> VectSharp.Canvas.AnimatedCanvas.CurrentFrameProperty  [static]
```

Initial value:

```
= AvaloniaProperty.Register<AnimatedCanvas, int>(nameof(CurrentFrame), coerce:  (ownerObject, val) =>
        {
            AnimatedCanvas owner = (AnimatedCanvas)ownerObject;
            if (owner.maxFrameIndex > 0)
            {
                if (val < owner.maxFrameIndex)
                {
                    return val;
                }
                else
                {
                    owner.IsPlaying = false;
                    return owner.maxFrameIndex - 1;
                }
            }
            else
            {
                return val % owner.RenderActions.Length;
            }
        })
```

Defines the CurrentFrame property.

Definition at line 33 of file AnimatedCanvas.cs.

7.3.3.2 FrameCountProperty

```
readonly DirectProperty<AnimatedCanvas, int> VectSharp.Canvas.AnimatedCanvas.FrameCount↩
Property = AvaloniaProperty.RegisterDirect<AnimatedCanvas, int>(nameof(FrameCount), o => o.↩
maxFrameIndex)  [static]
```

Defines the FrameCount property.

Definition at line 67 of file AnimatedCanvas.cs.

7.3.3.3 FrameRateProperty

```
readonly DirectProperty<AnimatedCanvas, double> VectSharp.Canvas.AnimatedCanvas.FrameRate↩
Property = AvaloniaProperty.RegisterDirect<AnimatedCanvas, double>(nameof(FrameRate), o =>
o.FrameRate)  [static]
```

Defines the FrameRate property.

Definition at line 91 of file AnimatedCanvas.cs.

7.3.3.4 IsPlayingProperty

```
readonly StyledProperty<bool> VectSharp.Canvas.AnimatedCanvas.IsPlayingProperty  [static]
```

Initial value:

```
= AvaloniaProperty.Register<AnimatedCanvas, bool>(nameof(IsPlaying), false, coerce:  (ownerObject, val) =>
        {
            AnimatedCanvas owner = (AnimatedCanvas)ownerObject;
            if (owner.IsPlaying && !val)
            {
                owner.RefreshTimer.Dispose();
            }
            else if (!owner.IsPlaying && val)
            {
                owner.RefreshTimer = new System.Threading.Timer(_ =>
                {
                    _ = Avalonia.Threading.Dispatcher.UIThread.InvokeAsync(() =>
                    {
                        owner.CurrentFrame++;
                    });
                }, null, 0, (long)(1000.0 / owner.FrameRate));
            }
            return val;
        })
```

Defines the IsPlaying property.

Definition at line 115 of file AnimatedCanvas.cs.

7.3.4 Property Documentation

7.3.4.1 CurrentFrame

```
int VectSharp.Canvas.AnimatedCanvas.CurrentFrame  [get], [set]
```

The current frame in the animation.

Definition at line 58 of file AnimatedCanvas.cs.

7.3.4.2 FrameCount

```
int VectSharp.Canvas.AnimatedCanvas.FrameCount  [get]
```

The number of frames in the animation.

Definition at line 74 of file AnimatedCanvas.cs.

7.3.4.3 FrameRate

```
double VectSharp.Canvas.AnimatedCanvas.FrameRate  [get]
```

The target frame rate of the animation.

Definition at line 98 of file AnimatedCanvas.cs.

7.3.4.4 IsPlaying

```
bool VectSharp.Canvas.AnimatedCanvas.IsPlaying  [get], [set]
```

The current frame in the animation.

Definition at line 140 of file AnimatedCanvas.cs.

The documentation for this class was generated from the following file:

- VectSharp.Canvas/AnimatedCanvas.cs

7.4 VectSharp.AnimatedPNG Class Reference

Contains methods to create animated PNG image files.

Classes

- class CompressedFrame

  Represents an individual frame of a PNG animation.

Public Types

- enum InterframeCompression

  Types of inter-frame compression.

Static Public Member Functions

- static unsafe void Create (Stream outputStream, int width, int height, bool hasAlpha, IReadOnlyList< CompressedFrame > compressedFrames, int repeatCount)

  Create a new animated PNG image, outputting it to the specified stream.

7.4.1 Detailed Description

Contains methods to create animated PNG image files.

Definition at line 30 of file ImageFormats.cs.

7.4.2 Member Enumeration Documentation

7.4.2.1 InterframeCompression

```
enum VectSharp.AnimatedPNG.InterframeCompression
```

Types of inter-frame compression.

Definition at line 35 of file ImageFormats.cs.

7.4.3 Member Function Documentation

7.4.3.1 Create()

```
static unsafe void VectSharp.AnimatedPNG.Create (
            Stream outputStream,
            int width,
            int height,
            bool hasAlpha,
            IReadOnlyList< CompressedFrame > compressedFrames,
            int repeatCount )  [static]
```

Create a new animated PNG image, outputting it to the specified stream.

Parameters

outputStream	The stream to which the animated PNG image will be written.
width	The width of the image in pixels.
height	The height of the image in pixels.
hasAlpha	If the frames of the image have an alpha channel, set this to `true`; otherwise, set it to `false`.
compressedFrames	The frames that will be used to create the animated PNG image.
repeatCount	The number of times that the animation should loop. Set this to 0 for an infinitely repeating animation.

Definition at line 181 of file ImageFormats.cs.

The documentation for this class was generated from the following file:

- VectSharp/ImageFormats.cs

7.5 VectSharp.Animation Class Reference

Describes an animation constituted by a number of frames and transitions between them.

Public Member Functions

- Animation (double width, double height, double linearisationResolution)

  Creates a new Animation with the specified width, height and linearisation resolution.
- Page GetFrameAtAbsolute (double time)

  Obtains the (interpolated) frame that should be displayed after the specified time has passed since the start of the animation.
- Page GetFrameAtRelative (double relativeTime)

  Obtains the (interpolated) frame that should be displayed after the specified relative time has passed since the start of the animation.
- void AddFrame (Frame frame, Transition transition=null)

  Adds a new frame to the animation, with the specified transition.
- void RemoveLastFrame ()

  Removes the last frame from the animation (and the corresponding transition).

Properties

- ImmutableList< Frame > Frames = ImmutableList<Frame>.Empty `[get]`

  The key frames of the animation.
- ImmutableList< Transition > Transitions = ImmutableList<Transition>.Empty `[get]`

  The transitions between successive frames of the animation. This array always contains one fewer element than Frames.
- double Width `[get, set]`

  The width of the animation.
- double Height `[get, set]`

  The height of the animation.
- Colour Background = Colour.FromRgba(255, 255, 255, 0) `[get, set]`

  The background colour of the animation.
- double LinearisationResolution `[get]`

  The absolute length between successive samples to use when linearising GraphicsPaths.
- double Duration = 0 `[get]`

  The total duration of the animation (not including the number of repeats).
- int RepeatCount = 0 `[get, set]`

  The number of times that the animation should repeat.

7.5.1 Detailed Description

Describes an animation constituted by a number of frames and transitions between them.

Definition at line 1410 of file Animation.cs.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 Animation()

```
VectSharp.Animation.Animation (
            double width,
            double height,
            double linearisationResolution )
```

Creates a new Animation with the specified width, height and linearisation resolution.

Parameters

width	The width of the animation.
height	The height of the animation.
linearisationResolution	The absolute length between successive samples to use when linearising GraphicsPaths.

Definition at line 1461 of file Animation.cs.

7.5.3 Member Function Documentation

7.5.3.1 AddFrame()

```
void VectSharp.Animation.AddFrame (
            Frame frame,
            Transition transition = null )
```

Adds a new frame to the animation, with the specified transition.

Parameters

frame	The new frame to add to the animation.
transition	The transition that should be applied between the previous frame and the new frame. This parameter is ignored for the first frame. If this is `null`, the animation will abruptly change from one frame to the next.

Definition at line 1586 of file Animation.cs.

7.5.3.2 GetFrameAtAbsolute()

```
Page VectSharp.Animation.GetFrameAtAbsolute (
            double time )
```

Obtains the (interpolated) frame that should be displayed after the specified time has passed since the start of the animation.

Parameters

time	The time since the start of the animation (in milliseconds).

Returns

A Page containing the interpolated frame that should be displayed after the specified time has passed since the start of the animation.

Definition at line 1473 of file Animation.cs.

7.5.3.3 GetFrameAtRelative()

```
Page VectSharp.Animation.GetFrameAtRelative (
            double relativeTime )
```

Obtains the (interpolated) frame that should be displayed after the specified relative time has passed since the start of the animation.

Parameters

relativeTime	The time since the start of the animation (ranging from 0 for the start of the animation, to 1 for the end of the animation).

Returns

A Page containing the interpolated frame that should be displayed after the specified time has passed since the start of the animation.

Definition at line 1576 of file Animation.cs.

7.5.3.4 RemoveLastFrame()

```
void VectSharp.Animation.RemoveLastFrame ( )
```

Removes the last frame from the animation (and the corresponding transition).

Exceptions

ArgumentOutOfRangeException	Thrown if there are no frames in the animation.

Definition at line 1608 of file Animation.cs.

7.5.4 Property Documentation

7.5.4.1 Background

```
Colour VectSharp.Animation.Background = Colour.FromRgba(255, 255, 255, 0)  [get], [set]
```

The background colour of the animation.

Definition at line 1438 of file Animation.cs.

7.5.4.2 Duration

```
double VectSharp.Animation.Duration = 0  [get]
```

The total duration of the animation (not including the number of repeats).

Definition at line 1448 of file Animation.cs.

7.5.4.3 Frames

```
ImmutableList<Frame> VectSharp.Animation.Frames = ImmutableList<Frame>.Empty  [get]
```

The key frames of the animation.

Definition at line 1415 of file Animation.cs.

7.5.4.4 Height

```
double VectSharp.Animation.Height  [get], [set]
```

The height of the animation.

Definition at line 1433 of file Animation.cs.

7.5.4.5 LinearisationResolution

```
double VectSharp.Animation.LinearisationResolution  [get]
```

The absolute length between successive samples to use when linearising GraphicsPaths.

Definition at line 1443 of file Animation.cs.

7.5.4.6 RepeatCount

```
int VectSharp.Animation.RepeatCount = 0  [get], [set]
```

The number of times that the animation should repeat.

Definition at line 1453 of file Animation.cs.

7.5.4.7 Transitions

```
ImmutableList<Transition> VectSharp.Animation.Transitions = ImmutableList<Transition>.Empty
[get]
```

The transitions between successive frames of the animation. This array always contains one fewer element than Frames.

Definition at line 1420 of file Animation.cs.

7.5.4.8 Width

```
double VectSharp.Animation.Width  [get], [set]
```

The width of the animation.

Definition at line 1428 of file Animation.cs.

The documentation for this class was generated from the following file:

- VectSharp/Animation.cs

7.6 VectSharp.PDF.AnnotationStyle Class Reference

Describes the appearance of a PDF annotation.

Public Member Functions

- AnnotationStyle (double borderWidth=1, LineDash? borderDash=null, Colour? borderColour=default)
  Create a new AnnotationStyle.

Properties

- double BorderWidth = 1 `[get, set]`
  Thickness of the annotation border.
- LineDash? BorderDash = null `[get, set]`
  Dash style for the annotation border.
- Colour BorderColour = Colours.Black `[get, set]`
  Colour for the annotation border.

7.6.1 Detailed Description

Describes the appearance of a PDF annotation.

Definition at line 26 of file AnnotationStyles.cs.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 AnnotationStyle()

```
VectSharp.PDF.AnnotationStyle.AnnotationStyle (
            double borderWidth = 1,
            LineDash?  borderDash = null,
            Colour?  borderColour = default )
```

Create a new AnnotationStyle.

Definition at line 47 of file AnnotationStyles.cs.

7.6.3 Property Documentation

7.6.3.1 BorderColour

```
Colour VectSharp.PDF.AnnotationStyle.BorderColour = Colours.Black  [get], [set]
```

Colour for the annotation border.

Definition at line 41 of file AnnotationStyles.cs.

7.6.3.2 BorderDash

```
LineDash?  VectSharp.PDF.AnnotationStyle.BorderDash = null  [get], [set]
```

Dash style for the annotation border.

Definition at line 36 of file AnnotationStyles.cs.

7.6.3.3 BorderWidth

```
double VectSharp.PDF.AnnotationStyle.BorderWidth = 1  [get], [set]
```

Thickness of the annotation border.

Definition at line 31 of file AnnotationStyles.cs.

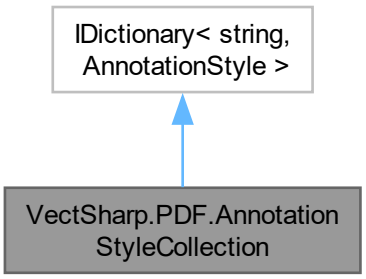
The documentation for this class was generated from the following file:

- VectSharp.PDF/AnnotationStyles.cs

7.7 VectSharp.PDF.AnnotationStyleCollection Class Reference

Represents annotation style settings.

Inheritance diagram for VectSharp.PDF.AnnotationStyleCollection:



Public Member Functions

- AnnotationStyleCollection (AnnotationStyle defaultStyle=null)

  Create a new set of AnnotationStyleCollection.
- void Add (string key, AnnotationStyle value)
- bool ContainsKey (string key)
- bool Remove (string key)
- bool TryGetValue (string key, out AnnotationStyle value)
- void Add (KeyValuePair< string, AnnotationStyle > item)
- void Clear ()
- bool Contains (KeyValuePair< string, AnnotationStyle > item)
- void CopyTo (KeyValuePair< string, AnnotationStyle >[] array, int arrayIndex)
- bool Remove (KeyValuePair< string, AnnotationStyle > item)
- IEnumerator< KeyValuePair< string, AnnotationStyle > > GetEnumerator ()

Properties

- AnnotationStyle DefaultStyle = new AnnotationStyle() `[get, set]`

  The default annotation style.
- Dictionary< string, AnnotationStyle > Styles `[get, set]`

  Annotation style overrides for individual annotations.
- ICollection< string > Keys `[get]`
- ICollection< AnnotationStyle > Values `[get]`
- int Count `[get]`
- bool IsReadOnly `[get]`
- AnnotationStyle this[string key] `[get, set]`

7.7.1 Detailed Description

Represents annotation style settings.

Definition at line 58 of file AnnotationStyles.cs.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 AnnotationStyleCollection()

```
VectSharp.PDF.AnnotationStyleCollection.AnnotationStyleCollection (
            AnnotationStyle defaultStyle = null )
```

Create a new set of AnnotationStyleCollection.

Parameters

defaultStyle	The default AnnotationStyle. If this is not provided or set to `null`, default values will be used.

Definition at line 89 of file AnnotationStyles.cs.

7.7.3 Member Function Documentation

7.7.3.1 Add() [1/2]

```
void VectSharp.PDF.AnnotationStyleCollection.Add (
            KeyValuePair< string, AnnotationStyle > item )
```

Definition at line 125 of file AnnotationStyles.cs.

7.7.3.2 Add() [2/2]

```
void VectSharp.PDF.AnnotationStyleCollection.Add (
            string key,
            AnnotationStyle value )
```

Definition at line 101 of file AnnotationStyles.cs.

7.7.3.3 Clear()

```
void VectSharp.PDF.AnnotationStyleCollection.Clear ( )
```

Definition at line 131 of file AnnotationStyles.cs.

7.7.3.4 Contains()

```
bool VectSharp.PDF.AnnotationStyleCollection.Contains (
            KeyValuePair< string, AnnotationStyle > item )
```

Definition at line 137 of file AnnotationStyles.cs.

7.7.3.5 ContainsKey()

```
bool VectSharp.PDF.AnnotationStyleCollection.ContainsKey (
            string key )
```

Definition at line 107 of file AnnotationStyles.cs.

7.7.3.6 CopyTo()

```
void VectSharp.PDF.AnnotationStyleCollection.CopyTo (
            KeyValuePair< string, AnnotationStyle >[] array,
            int arrayIndex )
```

Definition at line 143 of file AnnotationStyles.cs.

7.7.3.7 GetEnumerator()

```
IEnumerator< KeyValuePair< string, AnnotationStyle > > VectSharp.PDF.AnnotationStyleCollection.↩
GetEnumerator ( )
```

Definition at line 155 of file AnnotationStyles.cs.

7.7.3.8 Remove() [1/2]

```
bool VectSharp.PDF.AnnotationStyleCollection.Remove (
            KeyValuePair< string, AnnotationStyle > item )
```

Definition at line 149 of file AnnotationStyles.cs.

7.7.3.9 Remove() [2/2]

```
bool VectSharp.PDF.AnnotationStyleCollection.Remove (
            string key )
```

Definition at line 113 of file AnnotationStyles.cs.

7.7.3.10 TryGetValue()

```
bool VectSharp.PDF.AnnotationStyleCollection.TryGetValue (
            string key,
            out AnnotationStyle value )
```

Definition at line 119 of file AnnotationStyles.cs.

7.7.4 Property Documentation

7.7.4.1 Count

```
int VectSharp.PDF.AnnotationStyleCollection.Count  [get]
```

Definition at line 77 of file AnnotationStyles.cs.

7.7.4.2 DefaultStyle

```
AnnotationStyle VectSharp.PDF.AnnotationStyleCollection.DefaultStyle = new AnnotationStyle()
[get], [set]
```

The default annotation style.

Definition at line 63 of file AnnotationStyles.cs.

7.7.4.3 IsReadOnly

```
bool VectSharp.PDF.AnnotationStyleCollection.IsReadOnly  [get]
```

Definition at line 80 of file AnnotationStyles.cs.

7.7.4.4 Keys

```
ICollection<string> VectSharp.PDF.AnnotationStyleCollection.Keys [get]
```

Definition at line 71 of file AnnotationStyles.cs.

7.7.4.5 Styles

```
Dictionary<string, AnnotationStyle> VectSharp.PDF.AnnotationStyleCollection.Styles [get],
[set]
```

Annotation style overrides for individual annotations.

Definition at line 68 of file AnnotationStyles.cs.

7.7.4.6 this[string key]

```
AnnotationStyle VectSharp.PDF.AnnotationStyleCollection.this[string key] [get], [set]
```

Definition at line 83 of file AnnotationStyles.cs.

7.7.4.7 Values

```
ICollection<AnnotationStyle> VectSharp.PDF.AnnotationStyleCollection.Values [get]
```

Definition at line 74 of file AnnotationStyles.cs.

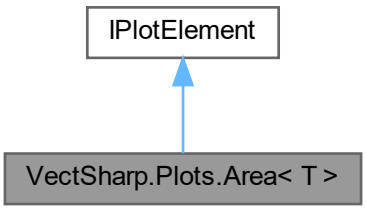
The documentation for this class was generated from the following file:

- VectSharp.PDF/AnnotationStyles.cs

7.8 VectSharp.Plots.Area< T > Class Template Reference

A plot element that fills an area between a line passing through some data points and a base line.

Inheritance diagram for VectSharp.Plots.Area< T >:



Public Member Functions

- Area (IEnumerable< T > data, Func< T, T > getBaseline, ICoordinateSystem< T > coordinateSystem)

  Create a new Area< T > instance.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IEnumerable< T > Data `[get, set]`

  The data points through which the upper part of the area will pass.
- Func< T, T > GetBaseline `[get, set]`

  A function returning the baseline for each data point.
- bool Smooth `[get, set]`

  If this is `false`, straight line segments are used to join the data points. If this is `true`, a smooth spline passing through all of them is used instead.
- ICoordinateSystem< T > CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() `[get, set]`

  Presentation attributes determining the appearance of the area.
- string Tag `[get, set]`

  A tag to identify the area in the plot.

7.8.1 Detailed Description

A plot element that fills an area between a line passing through some data points and a base line.

Template Parameters

T	The kind of data describing the data points (generally, `IReadOnlyList<double>`).

Definition at line 599 of file DataPoints.cs.

7.8.2 Constructor & Destructor Documentation

7.8.2.1 Area()

```
VectSharp.Plots.Area< T >.Area (
            IEnumerable< T > data,
```

```
Func< T, T > getBaseline,
ICoordinateSystem< T > coordinateSystem )
```

Create a new Area<T> instance.

Parameters

data	The data points through which the upper part of the area will pass.
getBaseline	A function returning the baseline for each data point.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 640 of file DataPoints.cs.

7.8.3 Member Function Documentation

7.8.3.1 Plot()

```
void VectSharp.Plots.Area< T >.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 648 of file DataPoints.cs.

7.8.4 Property Documentation

7.8.4.1 CoordinateSystem

```
ICoordinateSystem<T> VectSharp.Plots.Area< T >.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 621 of file DataPoints.cs.

7.8.4.2 Data

```
IEnumerable<T> VectSharp.Plots.Area< T >.Data  [get], [set]
```

The data points through which the upper part of the area will pass.

Definition at line 604 of file DataPoints.cs.

7.8.4.3 GetBaseline

```
Func<T, T> VectSharp.Plots.Area< T >.GetBaseline  [get], [set]
```

A function returning the baseline for each data point.

Definition at line 609 of file DataPoints.cs.

7.8.4.4 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.Area< T >.PresentationAttributes = new
PlotElementPresentationAttributes()  [get], [set]
```

Presentation attributes determining the appearance of the area.

Definition at line 627 of file DataPoints.cs.

7.8.4.5 Smooth

```
bool VectSharp.Plots.Area< T >.Smooth  [get], [set]
```

If this is `false`, straight line segments are used to join the data points. If this is `true`, a smooth spline passing through all of them is used instead.

Definition at line 615 of file DataPoints.cs.

7.8.4.6 Tag

```
string VectSharp.Plots.Area< T >.Tag  [get], [set]
```

A tag to identify the area in the plot.

Definition at line 632 of file DataPoints.cs.

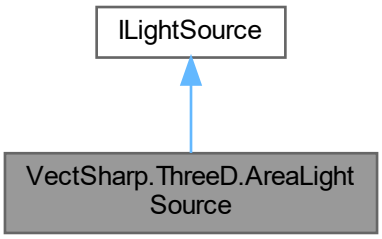
The documentation for this class was generated from the following file:

- VectSharp.Plots/DataPoints.cs

7.9 VectSharp.ThreeD.AreaLightSource Class Reference

Represents a light source emitting light from a circular area.

Inheritance diagram for VectSharp.ThreeD.AreaLightSource:



Public Member Functions

- AreaLightSource (double intensity, Point3D center, double radius, double penumbraRadius, Normalized↩ Vector3D direction, double sourceDistance, int shadowSamplingPointCount)

  Creates a new AreaLightSource instance.
- LightIntensity GetLightAt (Point3D point)

  Computes the light intensity at the specified point, without taking into account any obstructions.

Parameters

point	*The Point3DElement at which the light intensity should be computed.*

Returns

- double GetObstruction (Point3D point, IEnumerable< Triangle3DElement > shadowingTriangles)

  Determines the amount of obstruction of the light that results at point due to the specified shadowingTriangles .

Parameters

point	*The Point3D at which the obstruction should be computed.*
shadowingTriangles	*A collection of Triangle3DElement casting shadows.*

Returns

1 if the light is completely obstructed, 0 if the light is completely visible, a value between these if the light is partially obstructed.

Properties

- bool CastsShadow = true `[get, set]`

  Determines whether the light casts a shadow or not.
- Point3D Center `[get]`

The centre of the light-emitting area.

- NormalizedVector3D Direction `[get]`

  The direction of the light's main axis, i.e. the normal to the plane containing the light-emitting area.

- double Radius `[get]`

  The radius of the light emitting area.

- double PenumbraRadius `[get]`

  The radius of the penumbra area.

- double Intensity `[get, set]`

  The base intensity of the light.

- double SourceDistance `[get]`

  The distance between the focal point of the light and the light's Center.

- double DistanceAttenuationExponent = 2 `[get, set]`

  An exponent determining how fast the light attenuates with increasing distance. Set to 0 to disable distance attenuation.

- double PenumbraAttenuationExponent = 1 `[get, set]`

  An exponent determining how fast the light attenuates between the light-emitting area radius and the penumbra radius.

- int ShadowSamplingPointCount `[get]`

  The number of points to use when determining the amount of light that is obstructed at a certain point.

7.9.1 Detailed Description

Represents a light source emitting light from a circular area.

Definition at line 579 of file Lights.cs.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 AreaLightSource()

```
VectSharp.ThreeD.AreaLightSource.AreaLightSource (
            double intensity,
            Point3D center,
            double radius,
            double penumbraRadius,
            NormalizedVector3D direction,
            double sourceDistance,
            int shadowSamplingPointCount )
```

Creates a new AreaLightSource instance.

Parameters

intensity	The base intensity of the light.
center	The centre of the light-emitting area.
radius	The radius of the light-emitting area.
penumbraRadius	The radius of the penumbra area.
direction	The direction of the light.
sourceDistance	The distance between the focal point of the light and the light's center.
shadowSamplingPointCount	The number of points to use when determining the amount of light that is obstructed at a certain point.

Definition at line 643 of file Lights.cs.

7.9.3 Member Function Documentation

7.9.3.1 GetLightAt()

```
LightIntensity VectSharp.ThreeD.AreaLightSource.GetLightAt (
            Point3D point )
```

Computes the light intensity at the specified point, without taking into account any obstructions.

Parameters

point	The Point3DElement at which the light intensity should be computed.

Returns

Implements VectSharp.ThreeD.ILightSource.

Definition at line 690 of file Lights.cs.

7.9.3.2 GetObstruction()

```
double VectSharp.ThreeD.AreaLightSource.GetObstruction (
            Point3D point,
            IEnumerable< Triangle3DElement > shadowingTriangles )
```

Determines the amount of obstruction of the light that results at *point* due to the specified *shadowingTriangles* .

Parameters

point	The Point3D at which the obstruction should be computed.
shadowingTriangles	A collection of Triangle3DElement casting shadows.

Returns

1 if the light is completely obstructed, 0 if the light is completely visible, a value between these if the light is partially obstructed.

Implements VectSharp.ThreeD.ILightSource.

Definition at line 763 of file Lights.cs.

7.9.4 Property Documentation

7.9.4.1 CastsShadow

```
bool VectSharp.ThreeD.AreaLightSource.CastsShadow = true  [get], [set]
```

Determines whether the light casts a shadow or not.

Implements VectSharp.ThreeD.ILightSource.

Definition at line 582 of file Lights.cs.

7.9.4.2 Center

```
Point3D VectSharp.ThreeD.AreaLightSource.Center  [get]
```

The centre of the light-emitting area.

Definition at line 587 of file Lights.cs.

7.9.4.3 Direction

```
NormalizedVector3D VectSharp.ThreeD.AreaLightSource.Direction  [get]
```

The direction of the light's main axis, i.e. the normal to the plane containing the light-emitting area.

Definition at line 594 of file Lights.cs.

7.9.4.4 DistanceAttenuationExponent

```
double VectSharp.ThreeD.AreaLightSource.DistanceAttenuationExponent = 2  [get], [set]
```

An exponent determining how fast the light attenuates with increasing distance. Set to 0 to disable distance attenuation.

Definition at line 619 of file Lights.cs.

7.9.4.5 Intensity

```
double VectSharp.ThreeD.AreaLightSource.Intensity  [get], [set]
```

The base intensity of the light.

Definition at line 609 of file Lights.cs.

7.9.4.6 PenumbraAttenuationExponent

```
double VectSharp.ThreeD.AreaLightSource.PenumbraAttenuationExponent = 1  [get], [set]
```

An exponent determining how fast the light attenuates between the light-emitting area radius and the penumbra radius.

Definition at line 624 of file Lights.cs.

7.9.4.7 PenumbraRadius

```
double VectSharp.ThreeD.AreaLightSource.PenumbraRadius  [get]
```

The radius of the penumbra area.

Definition at line 604 of file Lights.cs.

7.9.4.8 Radius

```
double VectSharp.ThreeD.AreaLightSource.Radius  [get]
```

The radius of the light emitting area.

Definition at line 599 of file Lights.cs.

7.9.4.9 ShadowSamplingPointCount

```
int VectSharp.ThreeD.AreaLightSource.ShadowSamplingPointCount  [get]
```

The number of points to use when determining the amount of light that is obstructed at a certain point.

Definition at line 629 of file Lights.cs.

7.9.4.10 SourceDistance

```
double VectSharp.ThreeD.AreaLightSource.SourceDistance  [get]
```

The distance between the focal point of the light and the light's Center.

Definition at line 614 of file Lights.cs.

The documentation for this class was generated from the following file:

- VectSharp.ThreeD/Lights.cs

7.10 VectSharp.Canvas.AvaloniaContextInterpreter Class Reference

Contains methods to render a Page to an Avalonia.Controls.Canvas.

Public Types

- enum TextOptions

  Defines whether text items should be converted into paths when drawing.

Static Public Member Functions

- static Avalonia.Controls.Canvas PaintToCanvas (this Page page, TextOptions textOption=TextOptions.↩ ConvertIfNecessary, FilterOption filterOption=default)

  Render a Page to an Avalonia.Controls.Canvas.
- static Avalonia.Controls.Control PaintToCanvas (this Page page, bool graphicsAsControls, TextOptions text↩ Option=TextOptions.ConvertIfNecessary, FilterOption filterOption=default)

  Render a Page to an Avalonia.Controls.Control.
- static Avalonia.Controls.Control PaintToCanvas (this Page page, bool graphicsAsControls, Dictionary< string, Delegate > taggedActions, bool removeTaggedActionsAfterExecution=true, TextOptions textOption=Text↩ Options.ConvertIfNecessary, FilterOption filterOption=default)

  Render a Page to an Avalonia.Controls.Control.
- static Avalonia.Controls.Canvas PaintToCanvas (this Page page, Dictionary< string, Delegate > tagged↩ Actions, bool removeTaggedActionsAfterExecution=true, TextOptions textOption=TextOptions.ConvertIf↩ Necessary, FilterOption filterOption=default)

  Render a Page to an Avalonia.Controls.Control.
- static AnimatedCanvas PaintToAnimatedCanvas (this Animation animation, double frameRate=60, dou- ble durationScaling=1, TextOptions textOption=TextOptions.ConvertIfNecessary, FilterOption filter↩ Option=default)

  Render an Animation to an Avalonia.Controls.Control.

7.10.1 Detailed Description

Contains methods to render a Page to an Avalonia.Controls.Canvas.

Definition at line 2604 of file AvaloniaContext.cs.

7.10.2 Member Enumeration Documentation

7.10.2.1 TextOptions

```
enum VectSharp.Canvas.AvaloniaContextInterpreter.TextOptions
```

Defines whether text items should be converted into paths when drawing.

Definition at line 2609 of file AvaloniaContext.cs.

7.10.3 Member Function Documentation

7.10.3.1 PaintToAnimatedCanvas()

```
static AnimatedCanvas VectSharp.Canvas.AvaloniaContextInterpreter.PaintToAnimatedCanvas (
            this Animation animation,
            double frameRate = 60,
            double durationScaling = 1,
            TextOptions textOption = TextOptions.ConvertIfNecessary,
            FilterOption filterOption = default )  [static]
```

Render an Animation to an Avalonia.Controls.Control.

Parameters

animation	The Animation to render.
frameRate	The target frame rate of the animation, in frames-per-second (fps).
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
textOption	Defines whether text items should be converted into paths when drawing.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

An Avalonia.Controls.Control containing the rendered animation.

Definition at line 2727 of file AvaloniaContext.cs.

7.10.3.2 PaintToCanvas() [1/4]

```
static Avalonia.Controls.Control VectSharp.Canvas.AvaloniaContextInterpreter.PaintToCanvas (
            this Page page,
```

```
            bool graphicsAsControls,
            Dictionary< string, Delegate > taggedActions,
            bool removeTaggedActionsAfterExecution = true,
            TextOptions textOption = TextOptions.ConvertIfNecessary,
            FilterOption filterOption = default )  [static]
```

Render a Page to an Avalonia.Controls.Control.

Parameters

page	The Page to render.
graphicsAsControls	If this is true, each graphics object (e.g. paths, text...) is rendered as a separate Avalonia.Controls.Control. Otherwise, they are directly rendered onto the drawing context (which is faster, but does not allow interactivity).
taggedActions	A Dictionary<String, Delegate> containing the Actions that will be performed on items with the corresponding tag. If *graphicsAsControls* is true, the delegates should be voids that accept one parameter of type TextBlock or Path (depending on the tagged item), otherwise, they should accept one parameter of type RenderAction and return an IEnumerable<RenderAction> of the actions that will actually be performed.
removeTaggedActionsAfterExecution	Whether the Actions should be removed from *taggedActions* after their execution. Set to false if the same Action should be performed on multiple items with the same tag.
textOption	Defines whether text items should be converted into paths when drawing.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

An Avalonia.Controls.Control containing the rendered graphics objects.

Definition at line 2679 of file AvaloniaContext.cs.

7.10.3.3 PaintToCanvas() [2/4]

```
static Avalonia.Controls.Control VectSharp.Canvas.AvaloniaContextInterpreter.PaintToCanvas (
            this Page page,
            bool graphicsAsControls,
            TextOptions textOption = TextOptions.ConvertIfNecessary,
            FilterOption filterOption = default )  [static]
```

Render a Page to an Avalonia.Controls.Control.

Parameters

page	The Page to render.
graphicsAsControls	If this is true, each graphics object (e.g. paths, text...) is rendered as a separate Avalonia.Controls.Control. Otherwise, they are directly rendered onto the drawing context (which is faster, but does not allow interactivity).
textOption	Defines whether text items should be converted into paths when drawing
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

An Avalonia.Controls.Control containing the rendered graphics objects.

Definition at line 2652 of file AvaloniaContext.cs.

7.10.3.4 PaintToCanvas() [3/4]

```
static Avalonia.Controls.Canvas VectSharp.Canvas.AvaloniaContextInterpreter.PaintToCanvas (
            this Page page,
            Dictionary< string, Delegate > taggedActions,
            bool removeTaggedActionsAfterExecution = true,
            TextOptions textOption = TextOptions.ConvertIfNecessary,
            FilterOption filterOption = default )  [static]
```

Render a Page to an Avalonia.Controls.Control.

Parameters

page	The Page to render.
taggedActions	A Dictionary<String, Delegate> containing the Actions that will be performed on items with the corresponding tag. The delegates should accept one parameter of type TextBlock or Path (depending on the tagged item).
removeTaggedActionsAfterExecution	Whether the Actions should be removed from *taggedActions* after their execution. Set to false if the same Action should be performed on multiple items with the same tag.
textOption	Defines whether text items should be converted into paths when drawing.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

An Avalonia.Controls.Control containing the rendered graphics objects.

Definition at line 2705 of file AvaloniaContext.cs.

7.10.3.5 PaintToCanvas() [4/4]

```
static Avalonia.Controls.Canvas VectSharp.Canvas.AvaloniaContextInterpreter.PaintToCanvas (
            this Page page,
            TextOptions textOption = TextOptions.ConvertIfNecessary,
            FilterOption filterOption = default )  [static]
```

Render a Page to an Avalonia.Controls.Canvas.

Parameters

page	The Page to render.
textOption	Defines whether text items should be converted into paths when drawing.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

An Avalonia.Controls.Canvas containing the rendered graphics objects.

Definition at line 2634 of file AvaloniaContext.cs.

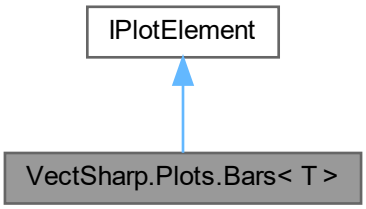
The documentation for this class was generated from the following file:

- VectSharp.Canvas/AvaloniaContext.cs

7.11 VectSharp.Plots.Bars< T > Class Template Reference

A plot element that draws bars.

Inheritance diagram for VectSharp.Plots.Bars< T >:



Public Member Functions

- Bars (IEnumerable< T > data, IComparer< T > sorting, Func< T, T > getBaseline, ICoordinateSystem< T > coordinateSystem)

  Create a new Bars<T> instance.
- Bars (IEnumerable< T > data, Comparison< T > sorting, Func< T, T > getBaseline, ICoordinateSystem< T > coordinateSystem)

  Create a new Bars<T> instance.
- Bars (IReadOnlyList< T > data, Func< T, T > getBaseline, ICoordinateSystem< T > coordinateSystem)

  Create a new Bars<T> instance.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

- Bars (IEnumerable< IReadOnlyList< double > > data, IComparer< IReadOnlyList< double > > sorting, Func< IReadOnlyList< double >, IReadOnlyList< double > > getBaseline, ICoordinateSystem< IRead↩ OnlyList< double > > coordinateSystem)

  Create a new Bars instance.
- Bars (IEnumerable< IReadOnlyList< double > > data, Comparison< IReadOnlyList< double > > sorting, Func< IReadOnlyList< double >, IReadOnlyList< double > > getBaseline, ICoordinateSystem< IRead↩ OnlyList< double > > coordinateSystem)

  Create a new Bars instance.
- Bars (IEnumerable< IReadOnlyList< double > > data, ICoordinateSystem< IReadOnlyList< double > > coordinateSystem, bool vertical=true)

  Create a new Bars instance.

Properties

- SortedSet< T > Data `[get, set]`

  The data points corresponding to the tips of the bars.
- Func< T, T > GetBaseline `[get, set]`

  A function that returns the bottom for each bar. This function should accept a single parameter of type T and return another T object, representing the bottom of the bar in data space.
- double Margin `[get, set]`

  The margin between consecutive bars. This should range between 0 and 1.
- ICoordinateSystem< T > CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() `[get, set]`

  Presentation attributes for the bars.
- string Tag `[get, set]`

  A tag to identify the bars in the plot.

7.11.1 Detailed Description

A plot element that draws bars.

A plot element that draws bars for numerical data.

Template Parameters

T	The type of data elements.

Definition at line 29 of file Bars.cs.

7.11.2 Constructor & Destructor Documentation

7.11.2.1 Bars() [1/6]

```
VectSharp.Plots.Bars< T >.Bars (
            IEnumerable< T > data,
            IComparer< T > sorting,
            Func< T, T > getBaseline,
            ICoordinateSystem< T > coordinateSystem )
```

Create a new Bars<T> instance.

Parameters

data	The data points corresponding to the tips of the bars.
sorting	A comparer used to sort the bars.
getBaseline	A function that returns the bottom for each bar. This function should accept a single parameter of type *T* and return another *T* object, representing the bottom of the bar in data space.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 89 of file Bars.cs.

7.11.2.2 Bars() [2/6]

```
VectSharp.Plots.Bars< T >.Bars (
            IEnumerable< T > data,
            Comparison< T > sorting,
            Func< T, T > getBaseline,
            ICoordinateSystem< T > coordinateSystem )
```

Create a new Bars<T> instance.

Parameters

data	The data points corresponding to the tips of the bars.
sorting	A comparer used to sort the bars.
getBaseline	A function that returns the bottom for each bar. This function should accept a single parameter of type *T* and return another *T* object, representing the bottom of the bar in data space.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 105 of file Bars.cs.

7.11.2.3 Bars() [3/6]

```
VectSharp.Plots.Bars< T >.Bars (
            IReadOnlyList< T > data,
```

```
            Func< T, T > getBaseline,
            ICoordinateSystem< T > coordinateSystem )
```

Create a new Bars<T> instance.

Parameters

data	The data points corresponding to the tips of the bars. These should already be sorted.
getBaseline	A function that returns the bottom for each bar. This function should accept a single parameter of type *T* and return another *T* object, representing the bottom of the bar in data space.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 115 of file Bars.cs.

7.11.2.4 Bars() [4/6]

```
VectSharp.Plots.Bars< T >.Bars (
            IEnumerable< IReadOnlyList< double > > data,
            IComparer< IReadOnlyList< double > > sorting,
            Func< IReadOnlyList< double >, IReadOnlyList< double > > getBaseline,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem )
```

Create a new Bars instance.

Parameters

data	The data points corresponding to the tips of the bars.
sorting	A comparer used to sort the bars.
getBaseline	A function that returns the bottom for each bar. This function should accept a single parameter (an IReadOnlyList<T> of doubles), and return another object of the same type, representing the bottom of the bar in data space.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 299 of file Bars.cs.

7.11.2.5 Bars() [5/6]

```
VectSharp.Plots.Bars< T >.Bars (
            IEnumerable< IReadOnlyList< double > > data,
            Comparison< IReadOnlyList< double > > sorting,
            Func< IReadOnlyList< double >, IReadOnlyList< double > > getBaseline,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem )
```

Create a new Bars instance.

Parameters

data	The data points corresponding to the tips of the bars.
sorting	A comparer used to sort the bars.
getBaseline	A function that returns the bottom for each bar. This function should accept a single parameter (an IReadOnlyList<T> of `double`s), and return another object of the same type, representing the bottom of the bar in data space.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 310 of file Bars.cs.

7.11.2.6 Bars() [6/6]

```
VectSharp.Plots.Bars< T >.Bars (
            IEnumerable< IReadOnlyList< double > > data,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem,
            bool vertical = true )
```

Create a new Bars instance.

Parameters

data	The data points corresponding to the tips of the bars.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.
vertical	If this is `true` (the default), the bars rise vertically above the X axis Otherwise, the bars grow horizontally from the Y axis.

Definition at line 319 of file Bars.cs.

7.11.3 Member Function Documentation

7.11.3.1 Plot()

```
void VectSharp.Plots.Bars< T >.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 143 of file Bars.cs.

7.11.4 Property Documentation

7.11.4.1 CoordinateSystem

```
ICoordinateSystem<T> VectSharp.Plots.Bars< T >.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 67 of file Bars.cs.

7.11.4.2 Data

```
SortedSet<T> VectSharp.Plots.Bars< T >.Data  [get], [set]
```

The data points corresponding to the tips of the bars.

Definition at line 36 of file Bars.cs.

7.11.4.3 GetBaseline

```
Func<T, T> VectSharp.Plots.Bars< T >.GetBaseline  [get], [set]
```

A function that returns the bottom for each bar. This function should accept a single parameter of type *T* and return another *T* object, representing the bottom of the bar in data space.

Definition at line 43 of file Bars.cs.

7.11.4.4 Margin

```
double VectSharp.Plots.Bars< T >.Margin  [get], [set]
```

The margin between consecutive bars. This should range between 0 and 1.

Definition at line 48 of file Bars.cs.

7.11.4.5 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.Bars< T >.PresentationAttributes = new
PlotElementPresentationAttributes()  [get], [set]
```

Presentation attributes for the bars.

Definition at line 73 of file Bars.cs.

7.11.4.6 Tag

```
string VectSharp.Plots.Bars< T >.Tag  [get], [set]
```

A tag to identify the bars in the plot.

Definition at line 78 of file Bars.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/Bars.cs

7.12 VectSharp.TrueTypeFile.Bearings Struct Reference

Represents the left- and right-side bearings of a glyph.

Public Attributes

- int LeftSideBearing

  The left-side bearing of the glyph.
- int RightSideBearing

  The right-side bearing of the glyph.

7.12.1 Detailed Description

Represents the left- and right-side bearings of a glyph.

Definition at line 2321 of file TrueType.cs.

7.12.2 Member Data Documentation

7.12.2.1 LeftSideBearing

```
int VectSharp.TrueTypeFile.Bearings.LeftSideBearing
```

The left-side bearing of the glyph.

Definition at line 2326 of file TrueType.cs.

7.12.2.2 RightSideBearing

```
int VectSharp.TrueTypeFile.Bearings.RightSideBearing
```

The right-side bearing of the glyph.

Definition at line 2331 of file TrueType.cs.

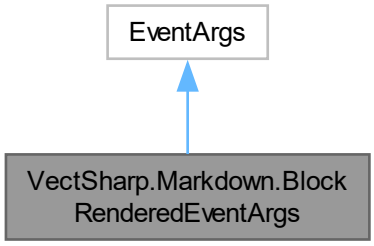
The documentation for this struct was generated from the following file:

- VectSharp/TrueType.cs

7.13 VectSharp.Markdown.BlockRenderedEventArgs Class Reference

EventArgs for the MarkdownRenderer.BlockRendered event.

Inheritance diagram for VectSharp.Markdown.BlockRenderedEventArgs:



Public Member Functions

- BlockRenderedEventArgs (MarkdownContext markdownContext, Graphics graphics, Block block)

  Create a new BlockRenderingEventArgs instance.

Properties

- MarkdownContext MarkdownContext `[get, set]`

  The current VectSharp.Markdown.MarkdownContext.
- Graphics Graphics `[get, set]`

  The current VectSharp.Graphics surface.
- Block Block `[get]`

  The Markdig.Syntax.Block that has been rendered.

7.13.1 Detailed Description

EventArgs for the MarkdownRenderer.BlockRendered event.

Definition at line 339 of file MarkdownRenderer.Events.cs.

7.13.2 Constructor & Destructor Documentation

7.13.2.1 BlockRenderedEventArgs()

```
VectSharp.Markdown.BlockRenderedEventArgs.BlockRenderedEventArgs (
            MarkdownContext markdownContext,
            Graphics graphics,
            Block block )
```

Create a new BlockRenderingEventArgs instance.

Parameters

markdownContext	The current VectSharp.Markdown.MarkdownContext.
graphics	The current VectSharp.Graphics surface.
block	The Markdig.Syntax.Block that has been rendered.

Definition at line 362 of file MarkdownRenderer.Events.cs.

7.13.3 Property Documentation

7.13.3.1 Block

```
Block VectSharp.Markdown.BlockRenderedEventArgs.Block  [get]
```

The Markdig.Syntax.Block that has been rendered.

Definition at line 354 of file MarkdownRenderer.Events.cs.

7.13.3.2 Graphics

```
Graphics VectSharp.Markdown.BlockRenderedEventArgs.Graphics  [get], [set]
```

The current VectSharp.Graphics surface.

Definition at line 349 of file MarkdownRenderer.Events.cs.

7.13.3.3 MarkdownContext

```
MarkdownContext VectSharp.Markdown.BlockRenderedEventArgs.MarkdownContext  [get], [set]
```

The current VectSharp.Markdown.MarkdownContext.

Definition at line 344 of file MarkdownRenderer.Events.cs.

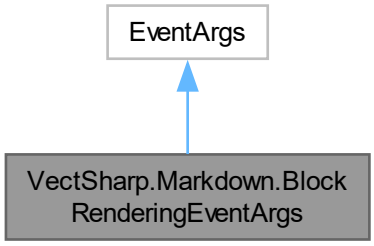
The documentation for this class was generated from the following file:

- VectSharp.Markdown/MarkdownRenderer.Events.cs

7.14 VectSharp.Markdown.BlockRenderingEventArgs Class Reference

EventArgs for the MarkdownRenderer.BlockRendering event.

Inheritance diagram for VectSharp.Markdown.BlockRenderingEventArgs:



Public Member Functions

- BlockRenderingEventArgs (MarkdownContext markdownContext, Graphics graphics, Block block)

  Create a new BlockRenderingEventArgs instance.

Properties

- MarkdownContext MarkdownContext `[get, set]`

  The current VectSharp.Markdown.MarkdownContext.
- Graphics Graphics `[get, set]`

  The VectSharp.Graphics surface on which the Block will be drawn.
- Block Block `[get, set]`

  The Markdig.Syntax.Block that will be rendered.

7.14.1 Detailed Description

EventArgs for the MarkdownRenderer.BlockRendering event.

Definition at line 305 of file MarkdownRenderer.Events.cs.

7.14.2 Constructor & Destructor Documentation

7.14.2.1 BlockRenderingEventArgs()

```
VectSharp.Markdown.BlockRenderingEventArgs.BlockRenderingEventArgs (
            MarkdownContext markdownContext,
            Graphics graphics,
            Block block )
```

Create a new BlockRenderingEventArgs instance.

Parameters

markdownContext	The current VectSharp.Markdown.MarkdownContext.
graphics	The VectSharp.Graphics surface on which the Block will be drawn.
block	The Markdig.Syntax.Block that will be rendered.

Definition at line 328 of file MarkdownRenderer.Events.cs.

7.14.3 Property Documentation

7.14.3.1 Block

```
Block VectSharp.Markdown.BlockRenderingEventArgs.Block  [get], [set]
```

The Markdig.Syntax.Block that will be rendered.

Definition at line 320 of file MarkdownRenderer.Events.cs.

7.14.3.2 Graphics

```
Graphics VectSharp.Markdown.BlockRenderingEventArgs.Graphics  [get], [set]
```

The VectSharp.Graphics surface on which the Block will be drawn.

Definition at line 315 of file MarkdownRenderer.Events.cs.

7.14.3.3 MarkdownContext

```
MarkdownContext VectSharp.Markdown.BlockRenderingEventArgs.MarkdownContext  [get], [set]
```

The current VectSharp.Markdown.MarkdownContext.

Definition at line 310 of file MarkdownRenderer.Events.cs.

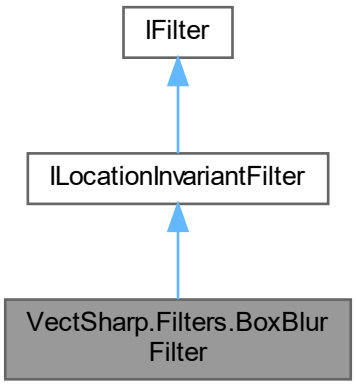
The documentation for this class was generated from the following file:

- VectSharp.Markdown/MarkdownRenderer.Events.cs

7.15 VectSharp.Filters.BoxBlurFilter Class Reference

Represents a filter applying a box blur.

Inheritance diagram for VectSharp.Filters.BoxBlurFilter:



Public Member Functions

- BoxBlurFilter (double boxRadius)

  Creates a new BoxBlurFilter with the specified radius.
- RasterImage Filter (RasterImage image, double scale)

  Applies the filter to a RasterImage.

Parameters

image	*The RasterImage to which the filter will be applied.*
scale	*The scale of the image with respect to the filter.*

Returns

A new RasterImage containing the filtered image. The source image is left unaltered.

Properties

- double BoxRadius `[get]`

  The radius of the box blur (the actual size of the box is 2 ∗ BoxRadius + 1).
- Point TopLeftMargin `[get]`

  Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.
- Point BottomRightMargin `[get]`

  Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.

7.15.1 Detailed Description

Represents a filter applying a box blur.

Definition at line 27 of file BoxBlurFilter.cs.

7.15.2 Constructor & Destructor Documentation

7.15.2.1 BoxBlurFilter()

```
VectSharp.Filters.BoxBlurFilter.BoxBlurFilter (
            double boxRadius )
```

Creates a new BoxBlurFilter with the specified radius.

Parameters

boxRadius	The radius of the box blur (the actual size of the box is 2 ∗ *boxRadius* + 1).

Definition at line 43 of file BoxBlurFilter.cs.

7.15.3 Member Function Documentation

7.15.3.1 Filter()

```
RasterImage VectSharp.Filters.BoxBlurFilter.Filter (
            RasterImage image,
            double scale )
```

Applies the filter to a RasterImage.

Parameters

image	The RasterImage to which the filter will be applied.
scale	The scale of the image with respect to the filter.

Returns

A new RasterImage containing the filtered image. The source *image* is left unaltered.

Implements VectSharp.Filters.ILocationInvariantFilter.

Definition at line 52 of file BoxBlurFilter.cs.

7.15.4 Property Documentation

7.15.4.1 BottomRightMargin

```
Point VectSharp.Filters.BoxBlurFilter.BottomRightMargin  [get]
```

Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.

Implements VectSharp.Filters.IFilter.

Definition at line 37 of file BoxBlurFilter.cs.

7.15.4.2 BoxRadius

```
double VectSharp.Filters.BoxBlurFilter.BoxRadius  [get]
```

The radius of the box blur (the actual size of the box is 2 ∗ BoxRadius + 1).

Definition at line 32 of file BoxBlurFilter.cs.

7.15.4.3 TopLeftMargin

```
Point VectSharp.Filters.BoxBlurFilter.TopLeftMargin  [get]
```

Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.

Implements VectSharp.Filters.IFilter.

Definition at line 35 of file BoxBlurFilter.cs.

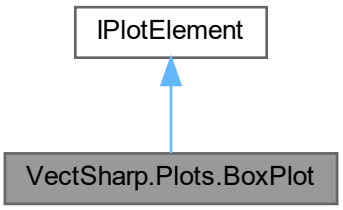
The documentation for this class was generated from the following file:

- VectSharp/Filters/BoxBlurFilter.cs

7.16 VectSharp.Plots.BoxPlot Class Reference

A plot element that draws a box plot.

Inheritance diagram for VectSharp.Plots.BoxPlot:



Public Member Functions

- BoxPlot (IReadOnlyList< double > position, IReadOnlyList< double > direction, double whisker1, double box1, double box2, double whisker2, ICoordinateSystem< IReadOnlyList< double > > coordinateSystem)

  Create a new BoxPlot instance.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IReadOnlyList< double > Position [get, set]

  The position of the centre (e.g., median or mean) of the box plot, in data space coordinates.
- IReadOnlyList< double > Direction [get, set]

  The direction along which the box plot is drawn, in data space coordinates.
- double Whisker1 [get, set]

  The distance between the centre of the box plot and the first whisker, expressed as a multiple of Direction.
- double Box1 [get, set]

  The distance between the centre of the box plot and the first side of the box, expressed as a multiple of Direction.
- double Box2 [get, set]

  The distance between the centre of the box plot and the second side of the box, expressed as a multiple of Direction.
- double Whisker2 [get, set]

  The distance between the centre of the box plot and the second whisker, expressed as a multiple of Direction.
- double Width = 10 [get, set]

  The width of the box plot, in data space coordinates.
- double WhiskerWidth = 0.5 [get, set]

  The width of the whiskers, expressed as a multiple of the Width of the box.
- double NotchWidth = 0.5 [get, set]

  The width of the notch, expressed as a multiple of the Width of the box.
- double NotchSize = 0 [get, set]

  The distance between the centre of the box plot and the end of the notch, expressed as a multiple of Direction.
- PlotElementPresentationAttributes BoxPresentationAttributes = new PlotElementPresentationAttributes() { Fill = Colours.White } [get, set]

  Presentation attributes for the box.
- PlotElementPresentationAttributes WhiskersPresentationAttributes = new PlotElementPresentationAttributes() { Fill = null } [get, set]

  Presentation attributes for the whiskers.
- PlotElementPresentationAttributes CentreSymbolPresentationAttributes = new PlotElementPresentationAttributes() { Fill = null, Stroke = null } [get, set]

  Presentation attributes for the symbol drawn at the centre of the box plot.
- IDataPointElement CentreSymbol = new PathDataPointElement() [get, set]

  Symbol drawn at the centre of the box plot.
- ICoordinateSystem< IReadOnlyList< double > > CoordinateSystem [get, set]

  The coordinate system used to transform the points from data space to plot space.
- string Tag [get, set]

  A tag to identify the box in the plot.

7.16.1 Detailed Description

A plot element that draws a box plot.

Definition at line 26 of file BoxPlot.cs.

7.16.2 Constructor & Destructor Documentation

7.16.2.1 BoxPlot()

```
VectSharp.Plots.BoxPlot.BoxPlot (
            IReadOnlyList< double > position,
            IReadOnlyList< double > direction,
            double whisker1,
            double box1,
            double box2,
            double whisker2,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem )
```

Create a new BoxPlot instance.

Parameters

position	The position of the centre (e.g., median or mean) of the box plot, in data space coordinates.
direction	The direction along which the box plot is drawn, in data space coordinates.
whisker1	The distance between the centre of the box plot and the first whisker, expressed as a multiple of *direction* .
box1	The distance between the centre of the box plot and the first side of the box, expressed as a multiple of *direction* .
box2	The distance between the centre of the box plot and the second side of the box, expressed as a multiple of *direction* .
whisker2	The distance between the centre of the box plot and the second whisker, expressed as a multiple of *direction* .
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 129 of file BoxPlot.cs.

7.16.3 Member Function Documentation

7.16.3.1 Plot()

```
void VectSharp.Plots.BoxPlot.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 141 of file BoxPlot.cs.

7.16.4 Property Documentation

7.16.4.1 Box1

```
double VectSharp.Plots.BoxPlot.Box1  [get], [set]
```

The distance between the centre of the box plot and the first side of the box, expressed as a multiple of Direction.

Definition at line 48 of file BoxPlot.cs.

7.16.4.2 Box2

```
double VectSharp.Plots.BoxPlot.Box2  [get], [set]
```

The distance between the centre of the box plot and the second side of the box, expressed as a multiple of Direction.

Definition at line 54 of file BoxPlot.cs.

7.16.4.3 BoxPresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.BoxPlot.BoxPresentationAttributes = new
PlotElementPresentationAttributes() { Fill = Colours.White }  [get], [set]
```

Presentation attributes for the box.

Definition at line 86 of file BoxPlot.cs.

7.16.4.4 CentreSymbol

```
IDataPointElement VectSharp.Plots.BoxPlot.CentreSymbol = new PathDataPointElement()  [get],
[set]
```

Symbol drawn at the centre of the box plot.

Definition at line 101 of file BoxPlot.cs.

7.16.4.5 CentreSymbolPresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.BoxPlot.CentreSymbolPresentationAttributes =
new PlotElementPresentationAttributes() { Fill = null, Stroke = null }  [get], [set]
```

Presentation attributes for the symbol drawn at the centre of the box plot.

Definition at line 96 of file BoxPlot.cs.

7.16.4.6 CoordinateSystem

```
ICoordinateSystem<IReadOnlyList<double> > VectSharp.Plots.BoxPlot.CoordinateSystem  [get],
[set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 106 of file BoxPlot.cs.

7.16.4.7 Direction

```
IReadOnlyList<double> VectSharp.Plots.BoxPlot.Direction  [get], [set]
```

The direction along which the box plot is drawn, in data space coordinates.

Definition at line 36 of file BoxPlot.cs.

7.16.4.8 NotchSize

```
double VectSharp.Plots.BoxPlot.NotchSize = 0  [get], [set]
```

The distance between the centre of the box plot and the end of the notch, expressed as a multiple of Direction.

Definition at line 81 of file BoxPlot.cs.

7.16.4.9 NotchWidth

```
double VectSharp.Plots.BoxPlot.NotchWidth = 0.5  [get], [set]
```

The width of the notch, expressed as a multiple of the Width of the box.

Definition at line 75 of file BoxPlot.cs.

7.16.4.10 Position

```
IReadOnlyList<double> VectSharp.Plots.BoxPlot.Position  [get], [set]
```

The position of the centre (e.g., median or mean) of the box plot, in data space coordinates.

Definition at line 31 of file BoxPlot.cs.

7.16.4.11 Tag

```
string VectSharp.Plots.BoxPlot.Tag  [get], [set]
```

A tag to identify the box in the plot.

Definition at line 111 of file BoxPlot.cs.

7.16.4.12 Whisker1

```
double VectSharp.Plots.BoxPlot.Whisker1  [get], [set]
```

The distance between the centre of the box plot and the first whisker, expressed as a multiple of Direction.

Definition at line 42 of file BoxPlot.cs.

7.16.4.13 Whisker2

```
double VectSharp.Plots.BoxPlot.Whisker2  [get], [set]
```

The distance between the centre of the box plot and the second whisker, expressed as a multiple of Direction.

Definition at line 60 of file BoxPlot.cs.

7.16.4.14 WhiskersPresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.BoxPlot.WhiskersPresentationAttributes = new
PlotElementPresentationAttributes() { Fill = null }  [get], [set]
```

Presentation attributes for the whiskers.

Definition at line 91 of file BoxPlot.cs.

7.16.4.15 WhiskerWidth

```
double VectSharp.Plots.BoxPlot.WhiskerWidth = 0.5  [get], [set]
```

The width of the whiskers, expressed as a multiple of the Width of the box.

Definition at line 70 of file BoxPlot.cs.

7.16.4.16 Width

```
double VectSharp.Plots.BoxPlot.Width = 10  [get], [set]
```

The width of the box plot, in data space coordinates.

Definition at line 65 of file BoxPlot.cs.

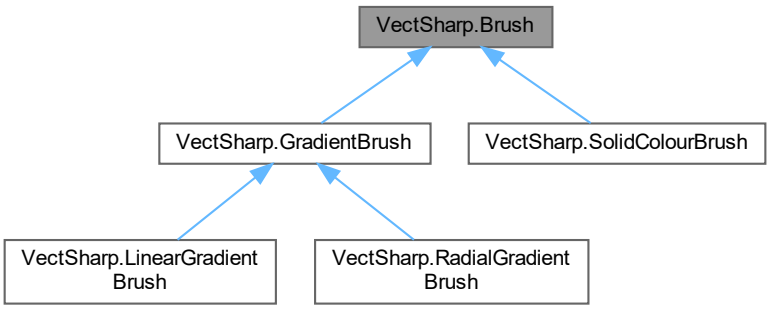
The documentation for this class was generated from the following file:

- VectSharp.Plots/BoxPlot.cs

7.17 VectSharp.Brush Class Reference

Represents a brush used to fill or stroke graphics elements. This could be a solid colour, or a more complicated gradient or pattern.

Inheritance diagram for VectSharp.Brush:



Public Member Functions

- abstract Brush MultiplyOpacity (double opacity)

  Returns a brush corresponding the current instance, with the specified opacity multiplication applied.

Static Public Member Functions

- static implicit operator Brush (Colour colour)

  Implicitly converts a Colour into a SolidColourBrush.

7.17.1 Detailed Description

Represents a brush used to fill or stroke graphics elements. This could be a solid colour, or a more complicated gradient or pattern.

Definition at line 31 of file Brush.cs.

7.17.2 Member Function Documentation

7.17.2.1 MultiplyOpacity()

```
abstract Brush VectSharp.Brush.MultiplyOpacity (
            double opacity )  [pure virtual]
```

Returns a brush corresponding the current instance, with the specified *opacity* multiplication applied.

Parameters

opacity	The value that will be used to multiply the opacity of the brush.

Returns

A brush corresponding the current instance, with the specified *opacity* multiplication applied.

Implemented in VectSharp.SolidColourBrush, VectSharp.LinearGradientBrush, and VectSharp.RadialGradientBrush.

7.17.2.2 operator Brush()

```
static implicit VectSharp.Brush.operator Brush (
            Colour colour )  [static]
```

Implicitly converts a Colour into a SolidColourBrush.

Parameters

colour	The Colour to use for the brush.

Definition at line 47 of file Brush.cs.

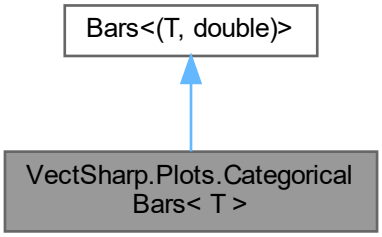
The documentation for this class was generated from the following file:

- VectSharp/Brush.cs

7.18 VectSharp.Plots.CategoricalBars< T > Class Template Reference

A plot element that draws bars for categorical data.

Inheritance diagram for VectSharp.Plots.CategoricalBars< T >:



Public Member Functions

- CategoricalBars (IEnumerable<(T, double)> data, IComparer<(T, double)> sorting, Func<(T, double),(T, double)> getBaseline, ICoordinateSystem<(T, double)> coordinateSystem)

  Create a new CategoricalBars<T> instance.
- CategoricalBars (IEnumerable<(T, double)> data, Comparison<(T, double)> sorting, Func<(T, double),(T, double)> getBaseline, ICoordinateSystem<(T, double)> coordinateSystem)

  Create a new CategoricalBars<T> instance.
- CategoricalBars (IEnumerable<(T, double)> data, Comparison<(T, double)> sorting, ICoordinateSystem<(T, double)> coordinateSystem)

  Create a new Bars<T> instance. The baseline for each `(T x, double y)` is determined automatically as `(x, 0)`.
- CategoricalBars (IReadOnlyList<(T, double)> data, ICoordinateSystem<(T, double)> coordinateSystem)

  Create a new Bars<T> instance. The baseline for each `(T x, double y)` is determined automatically as `(x, 0)`.

Additional Inherited Members

7.18.1 Detailed Description

A plot element that draws bars for categorical data.

Template Parameters

T	The type of the data categories.

Definition at line 245 of file Bars.cs.

7.18.2 Constructor & Destructor Documentation

7.18.2.1 CategoricalBars() [1/4]

```
VectSharp.Plots.CategoricalBars< T >.CategoricalBars (
            IEnumerable<(T, double)> data,
            IComparer<(T, double)> sorting,
            Func<(T, double),(T, double)> getBaseline,
            ICoordinateSystem<(T, double)> coordinateSystem )
```

Create a new CategoricalBars<T> instance.

Parameters

data	The data points corresponding to the tips of the bars.
sorting	A comparer used to sort the bars.
getBaseline	A function that returns the bottom for each bar. This function should accept a single parameter of type *T* and return another *T* object, representing the bottom of the bar in data space.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 256 of file Bars.cs.

7.18.2.2 CategoricalBars() [2/4]

```
VectSharp.Plots.CategoricalBars< T >.CategoricalBars (
            IEnumerable<(T, double)> data,
            Comparison<(T, double)> sorting,
            Func<(T, double),(T, double)> getBaseline,
            ICoordinateSystem<(T, double)> coordinateSystem )
```

Create a new CategoricalBars<T> instance.

Parameters

data	The data points corresponding to the tips of the bars.
sorting	A comparer used to sort the bars.
getBaseline	A function that returns the bottom for each bar. This function should accept a single parameter of type *T* and return another *T* object, representing the bottom of the bar in data space.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 267 of file Bars.cs.

7.18.2.3 CategoricalBars() [3/4]

```
VectSharp.Plots.CategoricalBars< T >.CategoricalBars (
            IEnumerable<(T, double)> data,
            Comparison<(T, double)> sorting,
            ICoordinateSystem<(T, double)> coordinateSystem )
```

Create a new Bars<T> instance. The baseline for each `(T x, double y)` is determined automatically as `(x, 0)`.

Parameters

data	The data points corresponding to the tips of the bars.
sorting	A comparer used to sort the bars.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 275 of file Bars.cs.

7.18.2.4 CategoricalBars() [4/4]

```
VectSharp.Plots.CategoricalBars< T >.CategoricalBars (
            IReadOnlyList<(T, double)> data,
            ICoordinateSystem<(T, double)> coordinateSystem )
```

Create a new Bars<T> instance. The baseline for each `(T x, double y)` is determined automatically as `(x, 0)`.

Parameters

data	The data points corresponding to the tips of the bars. These should already be sorted.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 282 of file Bars.cs.

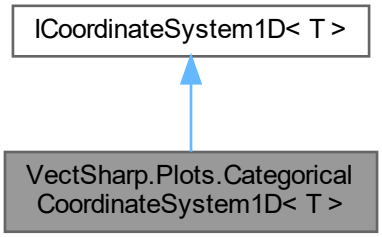
The documentation for this class was generated from the following file:

- VectSharp.Plots/Bars.cs

7.19 VectSharp.Plots.CategoricalCoordinateSystem1D< T > Class Template Reference

Represents a categorical 1-D coordinate system.

Inheritance diagram for VectSharp.Plots.CategoricalCoordinateSystem1D< T >:



Public Member Functions

- CategoricalCoordinateSystem1D (Dictionary< T, double > coordinates)

  Creates a new CategoricalCoordinateSystem1D<T> using the coordinates specified in the supplied coordinates Dictionary<TKey, TValue>.
- CategoricalCoordinateSystem1D (IReadOnlyList< T > data, double scale=350)

  Creates a new CategoricalCoordinateSystem1D<T> assigning equally-spaced coordinates to all different values of data .
- double ToPlotCoordinates (T dataPoint)

  Transform the specified dataPoint into a `double`*.*

Parameters

dataPoint	*The data point whose coordinates should be determined.*

Returns

A `double` *representing the dataPoint in plot space.*

Properties

- Dictionary< T, double > Coordinates `[get, set]`

  A Dictionary<TKey, TValue> storing the coordinates.

7.19.1 Detailed Description

Represents a categorical 1-D coordinate system.

Template Parameters

T	The type of data elements.

Definition at line 1083 of file CoordinateSystems.cs.

7.19.2 Constructor & Destructor Documentation

7.19.2.1 CategoricalCoordinateSystem1D() [1/2]

```
VectSharp.Plots.CategoricalCoordinateSystem1D< T >.CategoricalCoordinateSystem1D (
            Dictionary< T, double > coordinates )
```

Creates a new CategoricalCoordinateSystem1D<T> using the coordinates specified in the supplied *coordinates* Dictionary<TKey, TValue>.

Parameters

coordinates	

Definition at line 1094 of file CoordinateSystems.cs.

7.19.2.2 CategoricalCoordinateSystem1D() [2/2]

```
VectSharp.Plots.CategoricalCoordinateSystem1D< T >.CategoricalCoordinateSystem1D (
            IReadOnlyList< T > data,
            double scale = 350 )
```

Creates a new CategoricalCoordinateSystem1D<T> assigning equally-spaced coordinates to all different values of *data* .

Parameters

data	The data containing discrete parameter values that should be assigned to different coordinates.
scale	The maximum coordinate.

Definition at line 1104 of file CoordinateSystems.cs.

7.19.3 Member Function Documentation

7.19.3.1 ToPlotCoordinates()

```
double VectSharp.Plots.CategoricalCoordinateSystem1D< T >.ToPlotCoordinates (
            T dataPoint )
```

Transform the specified *dataPoint* into a `double`.

Parameters

dataPoint	The data point whose coordinates should be determined.

Returns

A `double` representing the *dataPoint* in plot space.

Implements VectSharp.Plots.ICoordinateSystem1D< T >.

Definition at line 1126 of file CoordinateSystems.cs.

7.19.4 Property Documentation

7.19.4.1 Coordinates

```
Dictionary<T, double> VectSharp.Plots.CategoricalCoordinateSystem1D< T >.Coordinates  [get],
[set]
```

A Dictionary<TKey, TValue> storing the coordinates.

Definition at line 1088 of file CoordinateSystems.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.20 VectSharp.TrueTypeFile.ClassDefinitionTable.ClassRangeRecord Struct Reference

Public Member Functions

- ClassRangeRecord (Stream fs)

Public Attributes

- ushort StartGlyphID
- ushort EndGlyphID
- ushort Class

7.20.1 Detailed Description

Definition at line 3927 of file TrueType.cs.

7.20.2 Constructor & Destructor Documentation

7.20.2.1 ClassRangeRecord()

```
VectSharp.TrueTypeFile.ClassDefinitionTable.ClassRangeRecord.ClassRangeRecord (
            Stream fs )
```

Definition at line 3933 of file TrueType.cs.

7.20.3 Member Data Documentation

7.20.3.1 Class

```
ushort VectSharp.TrueTypeFile.ClassDefinitionTable.ClassRangeRecord.Class
```

Definition at line 3931 of file TrueType.cs.

7.20.3.2 EndGlyphID

```
ushort VectSharp.TrueTypeFile.ClassDefinitionTable.ClassRangeRecord.EndGlyphID
```

Definition at line 3930 of file TrueType.cs.

7.20.3.3 StartGlyphID

```
ushort VectSharp.TrueTypeFile.ClassDefinitionTable.ClassRangeRecord.StartGlyphID
```

Definition at line 3929 of file TrueType.cs.

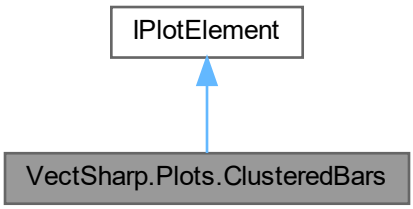
The documentation for this struct was generated from the following file:

- VectSharp/TrueType.cs

7.21 VectSharp.Plots.ClusteredBars Class Reference

A plot element that draws clusters of bars.

Inheritance diagram for VectSharp.Plots.ClusteredBars:



Public Member Functions

- ClusteredBars (IEnumerable< IReadOnlyList< double > > data, IComparer< IReadOnlyList< double > > sorting, Func< IReadOnlyList< double >, IReadOnlyList< double > > getBaseline, ICoordinateSystem< IReadOnlyList< double > > coordinateSystem)

  Create a new ClusteredBars instance.
- ClusteredBars (IEnumerable< IReadOnlyList< double > > data, Comparison< IReadOnlyList< double > > sorting, Func< IReadOnlyList< double >, IReadOnlyList< double > > getBaseline, ICoordinateSystem< IReadOnlyList< double > > coordinateSystem)

  Create a new ClusteredBars instance.
- ClusteredBars (IEnumerable< IReadOnlyList< double > > data, ICoordinateSystem< IReadOnlyList< double > > coordinateSystem, bool vertical=true)

  Create a new ClusteredBars instance.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- bool Vertical = true `[get, set]`

  If this is `true`, the bars rise vertically above the X axis Otherwise, the bars grow horizontally from the Y axis.
- SortedSet< IReadOnlyList< double > > Data `[get, set]`

  The data points corresponding to the tips of the bars. For each bar cluster, the data point contains an element determining the position of the cluster on the X axis (if Vertical is `true`, or on the Y axis otherwise), and a set of elements determining the length of each bar in the cluster.
- Func< IReadOnlyList< double >, IReadOnlyList< double > > GetBaseline `[get, set]`

A function that returns the bottom for each bar cluster. This function should accept a single parameter (an IReadOnlyList<T> of `doubles`), and return another object of the same type, representing the bottom of the cluster in data space.

- double InterClusterMargin `[get, set]`

  The margin between consecutive bar clusters.

- double IntraClusterMargin `[get, set]`

  The margin between consecutive bars within a single cluster.

- ICoordinateSystem< IReadOnlyList< double > > CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.

- IReadOnlyList< PlotElementPresentationAttributes > PresentationAttributes = new PlotElementPresentationAttributes[ ] { new PlotElementPresentationAttributes() } `[get, set]`

  Presentation attributes for the bars. An element from this collection is used for each bar in the cluster; if there are more bars than elements in this collection, the presentation attributes are wrapped.

- string Tag `[get, set]`

  A tag to identify the clustered bars in the plot.

7.21.1 Detailed Description

A plot element that draws clusters of bars.

Definition at line 625 of file Bars.cs.

7.21.2 Constructor & Destructor Documentation

7.21.2.1 ClusteredBars() [1/3]

```
VectSharp.Plots.ClusteredBars.ClusteredBars (
            IEnumerable< IReadOnlyList< double > > data,
            IComparer< IReadOnlyList< double > > sorting,
            Func< IReadOnlyList< double >, IReadOnlyList< double > > getBaseline,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem )
```

Create a new ClusteredBars instance.

Parameters

data	The data points corresponding to the tips of the bars. For each bar cluster, the data point contains an element determining the position of the cluster on the X axis (if Vertical is `true`, or on the Y axis otherwise), and a set of elements determining the length of each bar in the cluster.
sorting	A comparer used to sort the bar clusters.
getBaseline	A function that returns the bottom for each bar cluster. This function should accept a single parameter (an IReadOnlyList<T> of `doubles`), and return another object of the same type, representing the bottom of the cluster in data space.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 721 of file Bars.cs.

7.21.2.2 ClusteredBars() [2/3]

```
VectSharp.Plots.ClusteredBars.ClusteredBars (
            IEnumerable< IReadOnlyList< double > > data,
            Comparison< IReadOnlyList< double > > sorting,
            Func< IReadOnlyList< double >, IReadOnlyList< double > > getBaseline,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem )
```

Create a new ClusteredBars instance.

Parameters

data	The data points corresponding to the tips of the bars. For each bar cluster, the data point contains an element determining the position of the cluster on the X axis (if Vertical is `true`, or on the Y axis otherwise), and a set of elements determining the length of each bar in the cluster.
sorting	A comparer used to sort the bar clusters.
getBaseline	A function that returns the bottom for each bar cluster. This function should accept a single parameter (an IReadOnlyList<T> of `double`s), and return another object of the same type, representing the bottom of the cluster in data space.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 740 of file Bars.cs.

7.21.2.3 ClusteredBars() [3/3]

```
VectSharp.Plots.ClusteredBars.ClusteredBars (
            IEnumerable< IReadOnlyList< double > > data,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem,
            bool vertical = true )
```

Create a new ClusteredBars instance.

Parameters

data	The data points corresponding to the tips of the bars. For each bar cluster, the data point contains an element determining the position of the cluster on the X axis (if Vertical is `true`, or on the Y axis otherwise), and a set of elements determining the length of each bar in the cluster.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.
vertical	If this is `true` (the default), the bars rise vertically above the X axis Otherwise, the bars grow horizontally from the Y axis.

Definition at line 752 of file Bars.cs.

7.21.3 Member Function Documentation

7.21.3.1 Plot()

```
void VectSharp.Plots.ClusteredBars.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 755 of file Bars.cs.

7.21.4 Property Documentation

7.21.4.1 CoordinateSystem

```
ICoordinateSystem<IReadOnlyList<double> > VectSharp.Plots.ClusteredBars.CoordinateSystem
[get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 693 of file Bars.cs.

7.21.4.2 Data

```
SortedSet<IReadOnlyList<double> > VectSharp.Plots.ClusteredBars.Data  [get], [set]
```

The data points corresponding to the tips of the bars. For each bar cluster, the data point contains an element determining the position of the cluster on the X axis (if Vertical is `true`, or on the Y axis otherwise), and a set of elements determining the length of each bar in the cluster.

Definition at line 643 of file Bars.cs.

7.21.4.3 GetBaseline

```
Func<IReadOnlyList<double>, IReadOnlyList<double> > VectSharp.Plots.ClusteredBars.Get↩
Baseline  [get], [set]
```

A function that returns the bottom for each bar cluster. This function should accept a single parameter (an IRead↩ OnlyList<T> of doubles), and return another object of the same type, representing the bottom of the cluster in data space.

Definition at line 650 of file Bars.cs.

7.21.4.4 InterClusterMargin

```
double VectSharp.Plots.ClusteredBars.InterClusterMargin  [get], [set]
```

The margin between consecutive bar clusters.

Definition at line 655 of file Bars.cs.

7.21.4.5 IntraClusterMargin

```
double VectSharp.Plots.ClusteredBars.IntraClusterMargin  [get], [set]
```

The margin between consecutive bars within a single cluster.

Definition at line 674 of file Bars.cs.

7.21.4.6 PresentationAttributes

```
IReadOnlyList<PlotElementPresentationAttributes> VectSharp.Plots.ClusteredBars.Presentation↩
Attributes = new PlotElementPresentationAttributes[] { new PlotElementPresentationAttributes()
}  [get], [set]
```

Presentation attributes for the bars. An element from this collection is used for each bar in the cluster; if there are more bars than elements in this collection, the presentation attributes are wrapped.

Definition at line 701 of file Bars.cs.

7.21.4.7 Tag

```
string VectSharp.Plots.ClusteredBars.Tag  [get], [set]
```

A tag to identify the clustered bars in the plot.

Definition at line 706 of file Bars.cs.

7.21.4.8 Vertical

```
bool VectSharp.Plots.ClusteredBars.Vertical = true  [get], [set]
```

If this is `true`, the bars rise vertically above the X axis Otherwise, the bars grow horizontally from the Y axis.

Definition at line 634 of file Bars.cs.

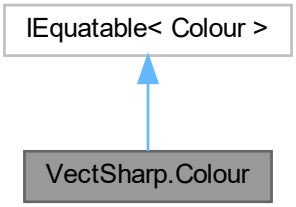
The documentation for this class was generated from the following file:

- VectSharp.Plots/Bars.cs

7.22 VectSharp.Colour Struct Reference

Represents an RGB colour.

Inheritance diagram for VectSharp.Colour:



Public Member Functions

- override bool Equals (object obj)
- bool Equals (Colour col)
- override int GetHashCode ()
- string ToCSSString (bool includeAlpha)

  Convert the Colour object into a hex string that is constituted by a "#" followed by two-digit hexadecimal representations of the red, green and blue components of the colour (in the range 0x00 - 0xFF). Optionally also includes opacity (alpha channel) data.
- Colour WithAlpha (double alpha)

  Create a new Colour with the same RGB components as the current Colour, but with the specified alpha .
- Colour WithAlpha (byte alpha)

  Create a new Colour with the same RGB components as the current Colour, but with the specified alpha .
- double double double Z ToXYZ ()
- double double double b ToLab ()
- double double double L ToHSL ()

Static Public Member Functions

- static Colour FromRgb (double r, double g, double b)

  Create a new colour from RGB (red, green and blue) values.
- static Colour FromRgb (byte r, byte g, byte b)

  Create a new colour from RGB (red, green and blue) values.
- static Colour FromRgb (int r, int g, int b)

  Create a new colour from RGB (red, green and blue) values.
- static Colour FromRgba (double r, double g, double b, double a)

  Create a new colour from RGBA (red, green, blue and alpha) values.
- static Colour FromRgba (byte r, byte g, byte b, byte a)

  Create a new colour from RGBA (red, green, blue and alpha) values.
- static Colour FromRgba (byte r, byte g, byte b, double a)

  Create a new colour from RGBA (red, green, blue and alpha) values.
- static Colour FromRgba (int r, int g, int b, int a)

  Create a new colour from RGBA (red, green, blue and alpha) values.
- static Colour FromRgba (int r, int g, int b, double a)

  Create a new colour from RGBA (red, green, blue and alpha) values.
- static Colour FromRgba ((int r, int g, int b, double a) colour)

  Create a new colour from RGBA (red, green, blue and alpha) values.
- static bool operator== (Colour col1, Colour col2)
- static bool operator!= (Colour col1, Colour col2)
- static ? Colour FromCSSString (string cssString)

  Convert a CSS colour string into a Colour object.
- static Colour WithAlpha (Colour original, double alpha)

  Create a new Colour with the same RGB components as the original Colour, but with the specified alpha .
- static Colour WithAlpha (Colour original, byte alpha)

  Create a new Colour with the same RGB components as the original Colour, but with the specified alpha .
- static Colour FromXYZ (double x, double y, double z)

  Creates a Colour from CIE XYZ coordinates.
- static Colour FromLab (double L, double a, double b)

  Creates a Colour from CIE Lab coordinates (under Illuminant D65).
- static Colour FromHSL (double h, double s, double l)

  Creates a Colour from HSL coordinates.

Public Attributes

- double R

  Red component of the colour. Range: [0, 1].
- double G

  Green component of the colour. Range: [0, 1].
- double B

  Blue component of the colour. Range: [0, 1].
- double A

  Alpha component of the colour. Range: [0, 1].
- double X

  Converts a Colour to the CIE XYZ colour space.
- double double Y
- double L

  Converts a Colour to the CIE Lab colour space (under Illuminant D65).
- double double a
- double H

  Converts a Colour to the HSL colour space.
- double double S

7.22.1 Detailed Description

Represents an RGB colour.

Definition at line 28 of file Colour.cs.

7.22.2 Member Function Documentation

7.22.2.1 Equals() [1/2]

```
bool VectSharp.Colour.Equals (
            Colour col )
```

Definition at line 182 of file Colour.cs.

7.22.2.2 Equals() [2/2]

```
override bool VectSharp.Colour.Equals (
            object obj )
```

Definition at line 169 of file Colour.cs.

7.22.2.3 FromCSSString()

```
static ?  Colour VectSharp.Colour.FromCSSString (
            string cssString )  [static]
```

Convert a CSS colour string into a Colour object.

Parameters

cssString	The CSS colour string. In addition to 148 standard colour names (case-insensitive), #RGB, #RGBA, #RRGGBB and #RRGGBBAA hex strings and rgb(r, g, b) and rgba(r, g, b, a) functional colour notations are supported.

Returns

Definition at line 228 of file Colour.cs.

7.22.2.4 FromHSL()

```
static Colour VectSharp.Colour.FromHSL (
            double h,
            double s,
            double l )  [static]
```

Creates a Colour from HSL coordinates.

Parameters

h	The H component. Should be in range [0, 1].
s	The S component. Should be in range [0, 1].
l	The L component. Should be in range [0, 1].

Returns

A Colour created from the specified components.

Definition at line 587 of file Colour.cs.

7.22.2.5 FromLab()

```
static Colour VectSharp.Colour.FromLab (
            double L,
            double a,
            double b )  [static]
```

Creates a Colour from CIE Lab coordinates (under Illuminant D65).

Parameters

L	The L∗ component.
a	The a∗ component.
b	The b∗ component.

Returns

An sRGB Colour created from the specified components.

Definition at line 509 of file Colour.cs.

7.22.2.6 FromRgb() [1/3]

```
static Colour VectSharp.Colour.FromRgb (
            byte r,
```

```
            byte g,
            byte b )  [static]
```

Create a new colour from RGB (red, green and blue) values.

Parameters

r	The red component of the colour. Range: [0, 255].
g	The green component of the colour. Range: [0, 255].
b	The blue component of the colour. Range: [0, 255].

Returns

A Colour struct with the specified components and an alpha component of 1.

Definition at line 77 of file Colour.cs.

7.22.2.7 FromRgb() [2/3]

```
static Colour VectSharp.Colour.FromRgb (
            double r,
            double g,
            double b )  [static]
```

Create a new colour from RGB (red, green and blue) values.

Parameters

r	The red component of the colour. Range: [0, 1].
g	The green component of the colour. Range: [0, 1].
b	The blue component of the colour. Range: [0, 1].

Returns

A Colour struct with the specified components and an alpha component of 1.

Definition at line 65 of file Colour.cs.

7.22.2.8 FromRgb() [3/3]

```
static Colour VectSharp.Colour.FromRgb (
            int r,
            int g,
            int b )  [static]
```

Create a new colour from RGB (red, green and blue) values.

Parameters

r	The red component of the colour. Range: [0, 255].
g	The green component of the colour. Range: [0, 255].
b	The blue component of the colour. Range: [0, 255].

Returns

A Colour struct with the specified components and an alpha component of 1.

Definition at line 89 of file Colour.cs.

7.22.2.9 FromRgba() [1/6]

```
static Colour VectSharp.Colour.FromRgba (
            (int r, int g, int b, double a) colour )  [static]
```

Create a new colour from RGBA (red, green, blue and alpha) values.

Parameters

colour	A ValueTuple<Int32, Int32, Int32, Double> containing component information for the colour. For r, g, and b, range: [0, 255]; for a, range: [0, 1].

Returns

A Colour struct with the specified components.

Definition at line 163 of file Colour.cs.

7.22.2.10 FromRgba() [2/6]

```
static Colour VectSharp.Colour.FromRgba (
            byte r,
            byte g,
            byte b,
            byte a )  [static]
```

Create a new colour from RGBA (red, green, blue and alpha) values.

Parameters

r	The red component of the colour. Range: [0, 255].
g	The green component of the colour. Range: [0, 255].
b	The blue component of the colour. Range: [0, 255].
a	The alpha component of the colour. Range: [0, 255].

Returns

A ColourColour struct with the specified components.

Definition at line 115 of file Colour.cs.

7.22.2.11 FromRgba() [3/6]

```
static Colour VectSharp.Colour.FromRgba (
            byte r,
            byte g,
            byte b,
            double a )  [static]
```

Create a new colour from RGBA (red, green, blue and alpha) values.

Parameters

r	The red component of the colour. Range: [0, 255].
g	The green component of the colour. Range: [0, 255].
b	The blue component of the colour. Range: [0, 255].
a	The alpha component of the colour. Range: [0, 1].

Returns

A Colour struct with the specified components.

Definition at line 128 of file Colour.cs.

7.22.2.12 FromRgba() [4/6]

```
static Colour VectSharp.Colour.FromRgba (
            double r,
            double g,
            double b,
            double a )  [static]
```

Create a new colour from RGBA (red, green, blue and alpha) values.

Parameters

r	The red component of the colour. Range: [0, 1].
g	The green component of the colour. Range: [0, 1].
b	The blue component of the colour. Range: [0, 1].
a	The alpha component of the colour. Range: [0, 1].

Returns

A Colour struct with the specified components.

Definition at line 102 of file Colour.cs.

7.22.2.13 FromRgba() [5/6]

```
static Colour VectSharp.Colour.FromRgba (
            int r,
            int g,
            int b,
            double a )  [static]
```

Create a new colour from RGBA (red, green, blue and alpha) values.

Parameters

r	The red component of the colour. Range: [0, 255].
g	The green component of the colour. Range: [0, 255].
b	The blue component of the colour. Range: [0, 255].
a	The alpha component of the colour. Range: [0, 1].

Returns

A Colour struct with the specified components.

Definition at line 153 of file Colour.cs.

7.22.2.14 FromRgba() [6/6]

```
static Colour VectSharp.Colour.FromRgba (
            int r,
            int g,
            int b,
            int a )  [static]
```

Create a new colour from RGBA (red, green, blue and alpha) values.

Parameters

r	The red component of the colour. Range: [0, 255].
g	The green component of the colour. Range: [0, 255].
b	The blue component of the colour. Range: [0, 255].
a	The alpha component of the colour. Range: [0, 255].

Returns

A Colour struct with the specified components.

Definition at line 140 of file Colour.cs.

7.22.2.15 FromXYZ()

```
static Colour VectSharp.Colour.FromXYZ (
            double x,
            double y,
            double z )  [static]
```

Creates a Colour from CIE XYZ coordinates.

Parameters

x	The X coordinate.
y	The Y coordinate.
z	The Z coordinate.

Returns

An sRGB Colour created from the specified components.

Definition at line 427 of file Colour.cs.

7.22.2.16 GetHashCode()

```
override int VectSharp.Colour.GetHashCode ( )
```

Definition at line 200 of file Colour.cs.

7.22.2.17 operator"!=()

```
static bool VectSharp.Colour.operator!= (
            Colour col1,
            Colour col2 )  [static]
```

Definition at line 194 of file Colour.cs.

7.22.2.18 operator==()

```
static bool VectSharp.Colour.operator== (
            Colour col1,
            Colour col2 )  [static]
```

Definition at line 188 of file Colour.cs.

7.22.2.19 ToCSSString()

```
string VectSharp.Colour.ToCSSString (
            bool includeAlpha )
```

Convert the Colour object into a hex string that is constituted by a "#" followed by two-digit hexadecimal representations of the red, green and blue components of the colour (in the range 0x00 - 0xFF). Optionally also includes opacity (alpha channel) data.

Parameters

includeAlpha	Whether two additional hex digits representing the colour's opacity (alpha channel) should be included in the string.

Returns

A hex colour string.

Definition at line 211 of file Colour.cs.

7.22.2.20 ToHSL()

```
double double double L VectSharp.Colour.ToHSL ( )
```

Definition at line 540 of file Colour.cs.

7.22.2.21 ToLab()

```
double double double b VectSharp.Colour.ToLab ( )
```

Definition at line 471 of file Colour.cs.

7.22.2.22 ToXYZ()

```
double double double Z VectSharp.Colour.ToXYZ ( )
```

Definition at line 382 of file Colour.cs.

7.22.2.23 WithAlpha() [1/4]

```
Colour VectSharp.Colour.WithAlpha (
            byte alpha )
```

Create a new Colour with the same RGB components as the current Colour, but with the specified *alpha* .

Parameters

alpha	The alpha component of the new Colour.

Returns

A Colour struct with the same RGB components as the current Colour and the specified *alpha* .

Definition at line 373 of file Colour.cs.

7.22.2.24 WithAlpha() [2/4]

```
static Colour VectSharp.Colour.WithAlpha (
            Colour original,
            byte alpha )  [static]
```

Create a new Colour with the same RGB components as the *original* Colour, but with the specified *alpha* .

Parameters

original	The original Colour from which the RGB components will be taken.
alpha	The alpha component of the new Colour.

Returns

A Colour struct with the same RGB components as the *original* Colour and the specified *alpha* .

Definition at line 351 of file Colour.cs.

7.22.2.25 WithAlpha() [3/4]

```
static Colour VectSharp.Colour.WithAlpha (
            Colour original,
            double alpha )  [static]
```

Create a new Colour with the same RGB components as the *original* Colour, but with the specified *alpha* .

Parameters

original	The original Colour from which the RGB components will be taken.
alpha	The alpha component of the new Colour.

Returns

A Colour struct with the same RGB components as the *original* Colour and the specified *alpha* .

Definition at line 340 of file Colour.cs.

7.22.2.26 WithAlpha() [4/4]

```
Colour VectSharp.Colour.WithAlpha (
            double alpha )
```

Create a new Colour with the same RGB components as the current Colour, but with the specified *alpha* .

Parameters

alpha	The alpha component of the new Colour.

Returns

A Colour struct with the same RGB components as the current Colour and the specified *alpha* .

Definition at line 362 of file Colour.cs.

7.22.3 Member Data Documentation

7.22.3.1 A

```
double VectSharp.Colour.A
```

Alpha component of the colour. Range: [0, 1].

Definition at line 48 of file Colour.cs.

7.22.3.2 a

```
double double VectSharp.Colour.a
```

Definition at line 471 of file Colour.cs.

7.22.3.3 B

```
double VectSharp.Colour.B
```

Blue component of the colour. Range: [0, 1].

Definition at line 43 of file Colour.cs.

7.22.3.4 G

```
double VectSharp.Colour.G
```

Green component of the colour. Range: [0, 1].

Definition at line 38 of file Colour.cs.

7.22.3.5 H

```
double VectSharp.Colour.H
```

Converts a Colour to the HSL colour space.

Returns

A ValueType containing the H, S and L components of the Colour. Each component has range [0, 1].

Definition at line 540 of file Colour.cs.

7.22.3.6 L

```
double VectSharp.Colour.L
```

Converts a Colour to the CIE Lab colour space (under Illuminant D65).

Returns

A ValueType containing the L$*$, a$*$ and b$*$ components of the Colour.

Definition at line 471 of file Colour.cs.

7.22.3.7 R

```
double VectSharp.Colour.R
```

Red component of the colour. Range: [0, 1].

Definition at line 33 of file Colour.cs.

7.22.3.8 S

```
double double VectSharp.Colour.S
```

Definition at line 540 of file Colour.cs.

7.22.3.9 X

```
double VectSharp.Colour.X
```

Converts a Colour to the CIE XYZ colour space.

Returns

A ValueTuple containing the X, Y and Z components of the Colour.

Definition at line 382 of file Colour.cs.

7.22.3.10 Y

```
double double VectSharp.Colour.Y
```

Definition at line 382 of file Colour.cs.

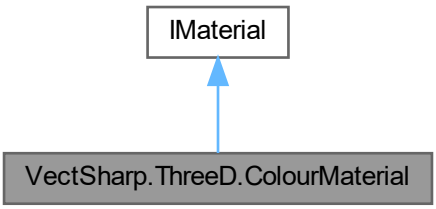
The documentation for this struct was generated from the following files:

- VectSharp/Colour.cs
- VectSharp/StandardColours.cs

7.23 VectSharp.ThreeD.ColourMaterial Class Reference

Represents a material that always has the same colour, regardless of light.

Inheritance diagram for VectSharp.ThreeD.ColourMaterial:



Public Member Functions

- ColourMaterial (Colour colour)

  Creates a new ColourMaterial instance.
- Colour GetColour (Point3D point, NormalizedVector3D surfaceNormal, Camera camera, IList< ILightSource > lights, IList< double > obstructions)

  Obtains the Colour at the specified point.

Parameters

point	*The point whose colour should be determined.*
surfaceNormal	*The normal to the surface at the specified point .*
camera	*The camera being used to render the scene.*
lights	*A list of light sources that are present in the scene.*
obstructions	*A list of values indicating how obstructed each light source is.*

Returns

The Colour of the specified point.

Properties

- Colour Colour `[get]`

  The colour of the material.

7.23.1 Detailed Description

Represents a material that always has the same colour, regardless of light.

Definition at line 48 of file Materials.cs.

7.23.2 Constructor & Destructor Documentation

7.23.2.1 ColourMaterial()

```
VectSharp.ThreeD.ColourMaterial.ColourMaterial (
            Colour colour )
```

Creates a new ColourMaterial instance.

Parameters

colour	The colour of the material.

Definition at line 59 of file Materials.cs.

7.23.3 Member Function Documentation

7.23.3.1 GetColour()

```
Colour VectSharp.ThreeD.ColourMaterial.GetColour (
            Point3D point,
            NormalizedVector3D surfaceNormal,
            Camera camera,
            IList< ILightSource > lights,
            IList< double > obstructions )
```

Obtains the Colour at the specified point.

Parameters

point	The point whose colour should be determined.
surfaceNormal	The normal to the surface at the specified *point* .
camera	The camera being used to render the scene.
lights	A list of light sources that are present in the scene.
obstructions	A list of values indicating how obstructed each light source is.

Returns

The Colour of the specified point.

Implements VectSharp.ThreeD.IMaterial.

Definition at line 65 of file Materials.cs.

7.23.4 Property Documentation

7.23.4.1 Colour

```
Colour VectSharp.ThreeD.ColourMaterial.Colour  [get]
```

The colour of the material.

Definition at line 53 of file Materials.cs.

The documentation for this class was generated from the following file:

- VectSharp.ThreeD/Materials.cs

7.24 VectSharp.Filters.ColourMatrix Class Reference

Represents a colour transformation matrix.

Public Member Functions

- ColourMatrix WithAlpha (double alpha)

  Creates a new ColourMatrix whose alpha coefficients are multiplied by the specified value.
- ColourMatrix (double[,] matrix)

  Creates a new ColourMatrix with the specified coefficients.
- Colour Apply (Colour colour)

  Applies the ColourMatrix to the specified Colour.
- void Apply (ref byte R, ref byte G, ref byte B, ref byte A)

  Applies the ColourMatrix to the specified colour, represented as four bytes, and stores the resulting colour in the same variables as the original RGBA values.
- void Apply (ref byte R, ref byte G, ref byte B)

  Applies the ColourMatrix to the specified colour, represented as three bytes, and stores the resulting colour in the same variables as the original RGB values.
- void Apply (byte R, byte G, byte B, byte A, out byte r, out byte g, out byte b, out byte a)

  Applies the ColourMatrix to the specified colour, represented as four bytes, and stores the resulting colour in the specified output bytes.
- void Apply (byte R, byte G, byte B, out byte r, out byte g, out byte b)

  Applies the ColourMatrix to the specified colour, represented as three bytes, and stores the resulting colour in the specified output bytes.

Static Public Member Functions

- static ColourMatrix ToColour (Colour colour, bool useAlpha=false)

  Creates a ColourMatrix that turns every colour to which it is applied into the specified colour .
- static ColourMatrix LuminanceToColour (Colour colour, bool useAlpha=false)

  Creates a ColourMatrix that turns every colour to which it is applied into a shade of the specified colour . The brightness of the output colour depends on the luminance of the input colour.
- static ColourMatrix LuminanceToAlpha (bool preserveColour=false)

  Creates a ColourMatrix that transforms the alpha value of the colour it is applied to into a value depending on the luminance of the input colour.
- static ColourMatrix operator∗ (ColourMatrix matrix1, ColourMatrix matrix2)

  Concatenates two matrices. The resulting ColourMatrix is equivalent to first applying matrix2 , and then matrix1 .

Static Public Attributes

- static ColourMatrix Identity = new ColourMatrix(new double[,] { { 1, 0, 0, 0, 0 }, { 0, 1, 0, 0, 0 }, { 0, 0, 1, 0, 0 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } })

  A ColourMatrix that whose output colour is always the same as the input colour.
- static ColourMatrix GreyScale = new ColourMatrix(new double[,] { { 0.2126, 0.7152, 0.0722, 0, 0 }, { 0.2126, 0.7152, 0.0722, 0, 0 }, { 0.2126, 0.7152, 0.0722, 0, 0 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } })

  A ColourMatrix that transforms every colour in a shade of grey with approximately the same luminance.
- static ColourMatrix Pastel = new ColourMatrix(new double[,] { { 0.75, 0.25, 0.25, 0, 0 }, { 0.25, 0.75, 0.25, 0, 0 }, { 0.25, 0.25, 0.75, 0, 0 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } })

  A ColourMatrix producing a "pastel" (desaturation) effect.
- static ColourMatrix Inversion = new ColourMatrix(new double[,] { { -1, 0, 0, 0, 1 }, { 0, -1, 0, 0, 1 }, { 0, 0, -1, 0, 1 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } })

  A ColourMatrix that inverts every colour, leaving the alpha component intact.
- static ColourMatrix AlphaInversion = new ColourMatrix(new double[,] { { 1, 0, 0, 0, 0 }, { 0, 1, 0, 0, 0 }, { 0, 0, 1, 0, 0 }, { 0, 0, 0, -1, 1 }, { 0, 0, 0, 0, 1 } })

  A ColourMatrix that inverts the alpha component, leaving the other components intact.
- static ColourMatrix InvertedAlphaShift = new ColourMatrix(new double[,] { { 1, 0, 0, -1, 1 }, { 0, 1, 0, -1, 1 }, { 0, 0, 1, -1, 1 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } })

  A ColourMatrix that shifts every colour component by an amount corresponding to the inverted alpha value. The alpha component is left intact.

Properties

- double R1 `[get, set]`

  The coefficient relating the R component of the output colour to the R component of the input colour.
- double R2 `[get, set]`

  The coefficient relating the R component of the output colour to the G component of the input colour.
- double R3 `[get, set]`

  The coefficient relating the R component of the output colour to the B component of the input colour.
- double R4 `[get, set]`

  The coefficient relating the R component of the output colour to the A component of the input colour.
- double R5 `[get, set]`

  The bias (translation) applied to the R component of the output colour.
- double G1 `[get, set]`

  The coefficient relating the G component of the output colour to the R component of the input colour.
- double G2 `[get, set]`

  The coefficient relating the G component of the output colour to the G component of the input colour.
- double G3 `[get, set]`

  The coefficient relating the G component of the output colour to the B component of the input colour.
- double G4 `[get, set]`

  The coefficient relating the G component of the output colour to the A component of the input colour.
- double G5 `[get, set]`

  The bias (translation) applied to the R component of the output colour.
- double B1 `[get, set]`

  The coefficient relating the B component of the output colour to the R component of the input colour.
- double B2 `[get, set]`

  The coefficient relating the B component of the output colour to the G component of the input colour.
- double B3 `[get, set]`

  The coefficient relating the B component of the output colour to the B component of the input colour.
- double B4 `[get, set]`

The coefficient relating the B component of the output colour to the A component of the input colour.

- double B5 `[get, set]`

  The bias (translation) applied to the B component of the output colour.

- double A1 `[get, set]`

  The coefficient relating the A component of the output colour to the R component of the input colour.

- double A2 `[get, set]`

  The coefficient relating the A component of the output colour to the G component of the input colour.

- double A3 `[get, set]`

  The coefficient relating the A component of the output colour to the B component of the input colour.

- double A4 `[get, set]`

  The coefficient relating the A component of the output colour to the A component of the input colour.

- double A5 `[get, set]`

  The bias (translation) applied to the A component of the output colour.

- double this[int y, int x] `[get]`

  Gets or sets the requested element of the matrix. Elements of the last row of the matrix can be read, but not set.

7.24.1 Detailed Description

Represents a colour transformation matrix.

Definition at line 27 of file ColourMatrixFilter.cs.

7.24.2 Constructor & Destructor Documentation

7.24.2.1 ColourMatrix()

```
VectSharp.Filters.ColourMatrix.ColourMatrix (
            double matrix[,] )
```

Creates a new ColourMatrix with the specified coefficients.

Parameters

matrix	The coefficients of the ColourMatrix.

Definition at line 456 of file ColourMatrixFilter.cs.

7.24.3 Member Function Documentation

7.24.3.1 Apply() [1/5]

```
void VectSharp.Filters.ColourMatrix.Apply (
            byte R,
            byte G,
            byte B,
            byte A,
            out byte r,
            out byte g,
            out byte b,
            out byte a )
```

Applies the ColourMatrix to the specified colour, represented as four bytes, and stores the resulting colour in the specified output bytes.

Parameters

R	The R component of the input colour.
G	The G component of the input colour.
B	The B component of the input colour.
A	The A component of the input colour.
r	After this method returns, this will contain the R component of the output colour.
g	After this method returns, this will contain the G component of the output colour.
b	After this method returns, this will contain the B component of the output colour.
a	After this method returns, this will contain the A component of the output colour.

Definition at line 541 of file ColourMatrixFilter.cs.

7.24.3.2 Apply() [2/5]

```
void VectSharp.Filters.ColourMatrix.Apply (
            byte R,
            byte G,
            byte B,
            out byte r,
            out byte g,
            out byte b )
```

Applies the ColourMatrix to the specified colour, represented as three bytes, and stores the resulting colour in the specified output bytes.

Parameters

R	The R component of the input colour.
G	The G component of the input colour.
B	The B component of the input colour.
r	After this method returns, this will contain the R component of the output colour.
g	After this method returns, this will contain the G component of the output colour.
b	After this method returns, this will contain the B component of the output colour.

Definition at line 559 of file ColourMatrixFilter.cs.

7.24.3.3 Apply() [3/5]

```
Colour VectSharp.Filters.ColourMatrix.Apply (
            Colour colour )
```

Applies the ColourMatrix to the specified Colour.

Parameters

colour	The Colour to which the ColourMatrix should be applied.

Returns

The result of applying the ColourMatrix to the specified colour.

Definition at line 473 of file ColourMatrixFilter.cs.

7.24.3.4 Apply() [4/5]

```
void VectSharp.Filters.ColourMatrix.Apply (
            ref byte R,
            ref byte G,
            ref byte B )
```

Applies the ColourMatrix to the specified colour, represented as three bytes, and stores the resulting colour in the same variables as the original RGB values.

Parameters

R	The R component of the input colour. After this method returns, this will contain the R component of the output colour.
G	The G component of the input colour. After this method returns, this will contain the G component of the output colour.
B	The B component of the input colour. After this method returns, this will contain the B component of the output colour.

Definition at line 518 of file ColourMatrixFilter.cs.

7.24.3.5 Apply() [5/5]

```
void VectSharp.Filters.ColourMatrix.Apply (
            ref byte R,
```

```
            ref byte G,
            ref byte B,
            ref byte A )
```

Applies the ColourMatrix to the specified colour, represented as four bytes, and stores the resulting colour in the same variables as the original RGBA values.

Parameters

R	The R component of the input colour. After this method returns, this will contain the R component of the output colour.
G	The G component of the input colour. After this method returns, this will contain the G component of the output colour.
B	The B component of the input colour. After this method returns, this will contain the B component of the output colour.
A	The A component of the input colour. After this method returns, this will contain the A component of the output colour.

Definition at line 498 of file ColourMatrixFilter.cs.

7.24.3.6 LuminanceToAlpha()

```
static ColourMatrix VectSharp.Filters.ColourMatrix.LuminanceToAlpha (
            bool preserveColour = false )  [static]
```

Creates a ColourMatrix that transforms the alpha value of the colour it is applied to into a value depending on the luminance of the input colour.

Parameters

preserveColour	If this is `true`, the values of the red, green and blue components of the input colour are preserved in the output colour. If this is false, the output colour will always be black.

Returns

A ColourMatrix that transforms the alpha value of the colour it is applied to into a value depending on the luminance of the input colour.

Definition at line 405 of file ColourMatrixFilter.cs.

7.24.3.7 LuminanceToColour()

```
static ColourMatrix VectSharp.Filters.ColourMatrix.LuminanceToColour (
            Colour colour,
            bool useAlpha = false )  [static]
```

Creates a ColourMatrix that turns every colour to which it is applied into a shade of the specified *colour* . The brightness of the output colour depends on the luminance of the input colour.

Parameters

colour	The colour whose shades will be produced by the ColourMatrix.
useAlpha	If this is `true`, the transformation will also be applied to the alpha channel. If this is false, the alpha value of the input pixels is preserved.

Returns

A ColourMatrix that turns every colour to which it is applied into a shade of the specified *colour* .

Definition at line 388 of file ColourMatrixFilter.cs.

7.24.3.8 operator∗()

```
static ColourMatrix VectSharp.Filters.ColourMatrix.operator* (
            ColourMatrix matrix1,
            ColourMatrix matrix2 )  [static]
```

Concatenates two matrices. The resulting ColourMatrix is equivalent to first applying *matrix2* , and then *matrix1* .

Parameters

matrix1	The matrix that acts second.
matrix2	The matrix that acts first.

Returns

A ColourMatrix equivalent to first applying *matrix2* , and then *matrix1* .

Definition at line 434 of file ColourMatrixFilter.cs.

7.24.3.9 ToColour()

```
static ColourMatrix VectSharp.Filters.ColourMatrix.ToColour (
            Colour colour,
            bool useAlpha = false )  [static]
```

Creates a ColourMatrix that turns every colour to which it is applied into the specified *colour* .

Parameters

colour	The colour that will be produced by the ColourMatrix.
useAlpha	If this is `true`, all output pixels will have the same alpha value as the supplied *colour* . If this is false, the alpha value of the input pixels is preserved.

Returns

A ColourMatrix that turns every colour to which it is applied into the specified *colour* .

Definition at line 370 of file ColourMatrixFilter.cs.

7.24.3.10 WithAlpha()

```
ColourMatrix VectSharp.Filters.ColourMatrix.WithAlpha (
            double alpha )
```

Creates a new ColourMatrix whose alpha coefficients are multiplied by the specified value.

Parameters

alpha	The value that will be used to multiply all the alpha coefficients of the ColourMatrix.

Returns

A new ColourMatrix whose alpha coefficients have been multiplied by the specified value.

Definition at line 423 of file ColourMatrixFilter.cs.

7.24.4 Member Data Documentation

7.24.4.1 AlphaInversion

```
ColourMatrix VectSharp.Filters.ColourMatrix.AlphaInversion = new ColourMatrix(new double[,] {
{ 1, 0, 0, 0, 0 }, { 0, 1, 0, 0, 0 }, { 0, 0, 1, 0, 0 }, { 0, 0, 0, -1, 1 }, { 0, 0, 0, 0, 1 }
})  [static]
```

A ColourMatrix that inverts the alpha component, leaving the other components intact.

Definition at line 357 of file ColourMatrixFilter.cs.

7.24.4.2 GreyScale

```
ColourMatrix VectSharp.Filters.ColourMatrix.GreyScale = new ColourMatrix(new double[,] { {
0.2126, 0.7152, 0.0722, 0, 0 }, { 0.2126, 0.7152, 0.0722, 0, 0 }, { 0.2126, 0.7152, 0.0722, 0,
0 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } })  [static]
```

A ColourMatrix that transforms every colour in a shade of grey with approximately the same luminance.

Definition at line 342 of file ColourMatrixFilter.cs.

7.24.4.3 Identity

```
ColourMatrix VectSharp.Filters.ColourMatrix.Identity = new ColourMatrix(new double[,] { { 1, 0, 0, 0, 0 }, { 0, 1, 0, 0, 0 }, { 0, 0, 1, 0, 0 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } })
[static]
```

A ColourMatrix that whose output colour is always the same as the input colour.

Definition at line 337 of file ColourMatrixFilter.cs.

7.24.4.4 Inversion

```
ColourMatrix VectSharp.Filters.ColourMatrix.Inversion = new ColourMatrix(new double[,] { { -1, 0, 0, 0, 1 }, { 0, -1, 0, 0, 1 }, { 0, 0, -1, 0, 1 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } })
[static]
```

A ColourMatrix that inverts every colour, leaving the alpha component intact.

Definition at line 352 of file ColourMatrixFilter.cs.

7.24.4.5 InvertedAlphaShift

```
ColourMatrix VectSharp.Filters.ColourMatrix.InvertedAlphaShift = new ColourMatrix(new double[,] { { 1, 0, 0, -1, 1 }, { 0, 1, 0, -1, 1 }, { 0, 0, 1, -1, 1 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } })  [static]
```

A ColourMatrix that shifts every colour component by an amount corresponding to the inverted alpha value. The alpha component is left intact.

Definition at line 362 of file ColourMatrixFilter.cs.

7.24.4.6 Pastel

```
ColourMatrix VectSharp.Filters.ColourMatrix.Pastel = new ColourMatrix(new double[,] { { 0.75, 0.25, 0.25, 0, 0 }, { 0.25, 0.75, 0.25, 0, 0 }, { 0.25, 0.25, 0.75, 0, 0 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } })  [static]
```

A ColourMatrix producing a "pastel" (desaturation) effect.

Definition at line 347 of file ColourMatrixFilter.cs.

7.24.5 Property Documentation

7.24.5.1 A1

```
double VectSharp.Filters.ColourMatrix.A1  [get], [set]
```

The coefficient relating the A component of the output colour to the R component of the input colour.

Definition at line 107 of file ColourMatrixFilter.cs.

7.24.5.2 A2

```
double VectSharp.Filters.ColourMatrix.A2  [get], [set]
```

The coefficient relating the A component of the output colour to the G component of the input colour.

Definition at line 112 of file ColourMatrixFilter.cs.

7.24.5.3 A3

```
double VectSharp.Filters.ColourMatrix.A3  [get], [set]
```

The coefficient relating the A component of the output colour to the B component of the input colour.

Definition at line 117 of file ColourMatrixFilter.cs.

7.24.5.4 A4

```
double VectSharp.Filters.ColourMatrix.A4  [get], [set]
```

The coefficient relating the A component of the output colour to the A component of the input colour.

Definition at line 122 of file ColourMatrixFilter.cs.

7.24.5.5 A5

```
double VectSharp.Filters.ColourMatrix.A5  [get], [set]
```

The bias (translation) applied to the A component of the output colour.

Definition at line 127 of file ColourMatrixFilter.cs.

7.24.5.6 B1

```
double VectSharp.Filters.ColourMatrix.B1  [get], [set]
```

The coefficient relating the B component of the output colour to the R component of the input colour.

Definition at line 82 of file ColourMatrixFilter.cs.

7.24.5.7 B2

```
double VectSharp.Filters.ColourMatrix.B2  [get], [set]
```

The coefficient relating the B component of the output colour to the G component of the input colour.

Definition at line 87 of file ColourMatrixFilter.cs.

7.24.5.8 B3

```
double VectSharp.Filters.ColourMatrix.B3  [get], [set]
```

The coefficient relating the B component of the output colour to the B component of the input colour.

Definition at line 92 of file ColourMatrixFilter.cs.

7.24.5.9 B4

```
double VectSharp.Filters.ColourMatrix.B4  [get], [set]
```

The coefficient relating the B component of the output colour to the A component of the input colour.

Definition at line 97 of file ColourMatrixFilter.cs.

7.24.5.10 B5

```
double VectSharp.Filters.ColourMatrix.B5  [get], [set]
```

The bias (translation) applied to the B component of the output colour.

Definition at line 102 of file ColourMatrixFilter.cs.

7.24.5.11 G1

```
double VectSharp.Filters.ColourMatrix.G1  [get], [set]
```

The coefficient relating the G component of the output colour to the R component of the input colour.

Definition at line 57 of file ColourMatrixFilter.cs.

7.24.5.12 G2

```
double VectSharp.Filters.ColourMatrix.G2  [get], [set]
```

The coefficient relating the G component of the output colour to the G component of the input colour.

Definition at line 62 of file ColourMatrixFilter.cs.

7.24.5.13 G3

```
double VectSharp.Filters.ColourMatrix.G3  [get], [set]
```

The coefficient relating the G component of the output colour to the B component of the input colour.

Definition at line 67 of file ColourMatrixFilter.cs.

7.24.5.14 G4

```
double VectSharp.Filters.ColourMatrix.G4  [get], [set]
```

The coefficient relating the G component of the output colour to the A component of the input colour.

Definition at line 72 of file ColourMatrixFilter.cs.

7.24.5.15 G5

```
double VectSharp.Filters.ColourMatrix.G5  [get], [set]
```

The bias (translation) applied to the R component of the output colour.

Definition at line 77 of file ColourMatrixFilter.cs.

7.24.5.16 R1

```
double VectSharp.Filters.ColourMatrix.R1  [get], [set]
```

The coefficient relating the R component of the output colour to the R component of the input colour.

Definition at line 32 of file ColourMatrixFilter.cs.

7.24.5.17 R2

```
double VectSharp.Filters.ColourMatrix.R2  [get], [set]
```

The coefficient relating the R component of the output colour to the G component of the input colour.

Definition at line 37 of file ColourMatrixFilter.cs.

7.24.5.18 R3

```
double VectSharp.Filters.ColourMatrix.R3  [get], [set]
```

The coefficient relating the R component of the output colour to the B component of the input colour.

Definition at line 42 of file ColourMatrixFilter.cs.

7.24.5.19 R4

```
double VectSharp.Filters.ColourMatrix.R4  [get], [set]
```

The coefficient relating the R component of the output colour to the A component of the input colour.

Definition at line 47 of file ColourMatrixFilter.cs.

7.24.5.20 R5

```
double VectSharp.Filters.ColourMatrix.R5  [get], [set]
```

The bias (translation) applied to the R component of the output colour.

Definition at line 52 of file ColourMatrixFilter.cs.

7.24.5.21 this[int y, int x]

```
double VectSharp.Filters.ColourMatrix.this[int y, int x]  [get]
```

Gets or sets the requested element of the matrix. Elements of the last row of the matrix can be read, but not set.

Parameters

y	The row of the matrix.
x	The column of the matrix.

Returns

The requested element of the matrix.

Exceptions

ArgumentOutOfRangeException	An attempt has been made to access an element out of the bounds of the matrix.

Definition at line 136 of file ColourMatrixFilter.cs.

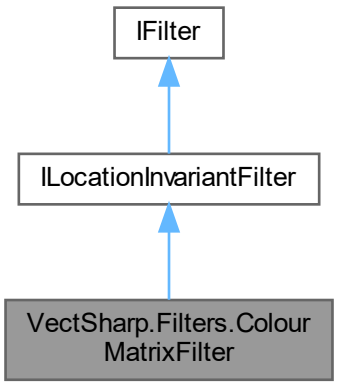
The documentation for this class was generated from the following file:

- VectSharp/Filters/ColourMatrixFilter.cs

7.25 VectSharp.Filters.ColourMatrixFilter Class Reference

Represents a filter that applies a Filters.ColourMatrix to the colours of the image.

Inheritance diagram for VectSharp.Filters.ColourMatrixFilter:



Public Member Functions

- ColourMatrixFilter (ColourMatrix colorMatrix)

  Creates a new ColourMatrixFilter with the specified Filters.ColourMatrix.
- RasterImage Filter (RasterImage image, double scale)

  Applies the filter to a RasterImage.

Parameters

image	*The RasterImage to which the filter will be applied.*
scale	*The scale of the image with respect to the filter.*

Returns

A new RasterImage containing the filtered image. The source image is left unaltered.

Properties

- ColourMatrix ColourMatrix `[get]`

  The Filters.ColourMatrix that is applied by this filter.
- Point TopLeftMargin = new Point() `[get]`

  Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.
- Point BottomRightMargin = new Point() `[get]`

  Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.

7.25.1 Detailed Description

Represents a filter that applies a Filters.ColourMatrix to the colours of the image.

Definition at line 570 of file ColourMatrixFilter.cs.

7.25.2 Constructor & Destructor Documentation

7.25.2.1 ColourMatrixFilter()

```
VectSharp.Filters.ColourMatrixFilter.ColourMatrixFilter (
            ColourMatrix colorMatrix )
```

Creates a new ColourMatrixFilter with the specified Filters.ColourMatrix.

Parameters

colorMatrix	The Filters.ColourMatrix that will be applied by the filter.

Definition at line 587 of file ColourMatrixFilter.cs.

7.25.3 Member Function Documentation

7.25.3.1 Filter()

```
RasterImage VectSharp.Filters.ColourMatrixFilter.Filter (
            RasterImage image,
            double scale )
```

Applies the filter to a RasterImage.

Parameters

image	The RasterImage to which the filter will be applied.
scale	The scale of the image with respect to the filter.

Returns

A new RasterImage containing the filtered image. The source *image* is left unaltered.

Implements VectSharp.Filters.ILocationInvariantFilter.

Definition at line 594 of file ColourMatrixFilter.cs.

7.25.4 Property Documentation

7.25.4.1 BottomRightMargin

```
Point VectSharp.Filters.ColourMatrixFilter.BottomRightMargin = new Point()  [get]
```

Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.

Implements VectSharp.Filters.IFilter.

Definition at line 581 of file ColourMatrixFilter.cs.

7.25.4.2 ColourMatrix

```
ColourMatrix VectSharp.Filters.ColourMatrixFilter.ColourMatrix  [get]
```

The Filters.ColourMatrix that is applied by this filter.

Definition at line 575 of file ColourMatrixFilter.cs.

7.25.4.3 TopLeftMargin

```
Point VectSharp.Filters.ColourMatrixFilter.TopLeftMargin = new Point()  [get]
```

Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.

Implements VectSharp.Filters.IFilter.

Definition at line 578 of file ColourMatrixFilter.cs.

The documentation for this class was generated from the following file:

- VectSharp/Filters/ColourMatrixFilter.cs

7.26 VectSharp.Colours Class Reference

Standard colours.

Static Public Attributes

- static Colour Black = Colour.FromRgb(0, 0, 0)
  Black #000000
- static Colour Navy = Colour.FromRgb(0, 0, 128)
  Navy #000080
- static Colour DarkBlue = Colour.FromRgb(0, 0, 139)
  DarkBlue #00008B
- static Colour MediumBlue = Colour.FromRgb(0, 0, 205)
  MediumBlue #0000CD
- static Colour Blue = Colour.FromRgb(0, 0, 255)
  Blue #0000FF
- static Colour DarkGreen = Colour.FromRgb(0, 100, 0)
  DarkGreen #006400
- static Colour Green = Colour.FromRgb(0, 128, 0)
  Green #008000
- static Colour Teal = Colour.FromRgb(0, 128, 128)
  Teal #008080
- static Colour DarkCyan = Colour.FromRgb(0, 139, 139)
  DarkCyan #008B8B
- static Colour DeepSkyBlue = Colour.FromRgb(0, 191, 255)
  DeepSkyBlue #00BFFF
- static Colour DarkTurquoise = Colour.FromRgb(0, 206, 209)
  DarkTurquoise #00CED1
- static Colour MediumSpringGreen = Colour.FromRgb(0, 250, 154)
  MediumSpringGreen #00FA9A
- static Colour Lime = Colour.FromRgb(0, 255, 0)
  Lime #00FF00
- static Colour SpringGreen = Colour.FromRgb(0, 255, 127)
  SpringGreen #00FF7F

- static Colour Aqua = Colour.FromRgb(0, 255, 255)

  Aqua #00FFFF
- static Colour Cyan = Colour.FromRgb(0, 255, 255)

  Cyan #00FFFF
- static Colour MidnightBlue = Colour.FromRgb(25, 25, 112)

  MidnightBlue #191970
- static Colour DodgerBlue = Colour.FromRgb(30, 144, 255)

  DodgerBlue #1E90FF
- static Colour LightSeaGreen = Colour.FromRgb(32, 178, 170)

  LightSeaGreen #20B2AA
- static Colour ForestGreen = Colour.FromRgb(34, 139, 34)

  ForestGreen #228B22
- static Colour SeaGreen = Colour.FromRgb(46, 139, 87)

  SeaGreen #2E8B57
- static Colour DarkSlateGray = Colour.FromRgb(47, 79, 79)

  DarkSlateGray #2F4F4F
- static Colour DarkSlateGrey = Colour.FromRgb(47, 79, 79)

  DarkSlateGrey #2F4F4F
- static Colour LimeGreen = Colour.FromRgb(50, 205, 50)

  LimeGreen #32CD32
- static Colour MediumSeaGreen = Colour.FromRgb(60, 179, 113)

  MediumSeaGreen #3CB371
- static Colour Turquoise = Colour.FromRgb(64, 224, 208)

  Turquoise #40E0D0
- static Colour RoyalBlue = Colour.FromRgb(65, 105, 225)

  RoyalBlue #4169E1
- static Colour SteelBlue = Colour.FromRgb(70, 130, 180)

  SteelBlue #4682B4
- static Colour DarkSlateBlue = Colour.FromRgb(72, 61, 139)

  DarkSlateBlue #483D8B
- static Colour MediumTurquoise = Colour.FromRgb(72, 209, 204)

  MediumTurquoise #48D1CC
- static Colour Indigo = Colour.FromRgb(75, 0, 130)

  Indigo #4B0082
- static Colour DarkOliveGreen = Colour.FromRgb(85, 107, 47)

  DarkOliveGreen #556B2F
- static Colour CadetBlue = Colour.FromRgb(95, 158, 160)

  CadetBlue #5F9EA0
- static Colour CornflowerBlue = Colour.FromRgb(100, 149, 237)

  CornflowerBlue #6495ED
- static Colour RebeccaPurple = Colour.FromRgb(102, 51, 153)

  RebeccaPurple #663399
- static Colour MediumAquaMarine = Colour.FromRgb(102, 205, 170)

  MediumAquaMarine #66CDAA
- static Colour DimGray = Colour.FromRgb(105, 105, 105)

  DimGray #696969
- static Colour DimGrey = Colour.FromRgb(105, 105, 105)

  DimGrey #696969
- static Colour SlateBlue = Colour.FromRgb(106, 90, 205)

  SlateBlue #6A5ACD
- static Colour OliveDrab = Colour.FromRgb(107, 142, 35)

OliveDrab #6B8E23

- static Colour SlateGray = Colour.FromRgb(112, 128, 144)

  SlateGray #708090
- static Colour SlateGrey = Colour.FromRgb(112, 128, 144)

  SlateGrey #708090
- static Colour LightSlateGray = Colour.FromRgb(119, 136, 153)

  LightSlateGray #778899
- static Colour LightSlateGrey = Colour.FromRgb(119, 136, 153)

  LightSlateGrey #778899
- static Colour MediumSlateBlue = Colour.FromRgb(123, 104, 238)

  MediumSlateBlue #7B68EE
- static Colour LawnGreen = Colour.FromRgb(124, 252, 0)

  LawnGreen #7CFC00
- static Colour Chartreuse = Colour.FromRgb(127, 255, 0)

  Chartreuse #7FFF00
- static Colour Aquamarine = Colour.FromRgb(127, 255, 212)

  Aquamarine #7FFFD4
- static Colour Maroon = Colour.FromRgb(128, 0, 0)

  Maroon #800000
- static Colour Purple = Colour.FromRgb(128, 0, 128)

  Purple #800080
- static Colour Olive = Colour.FromRgb(128, 128, 0)

  Olive #808000
- static Colour Gray = Colour.FromRgb(128, 128, 128)

  Gray #808080
- static Colour Grey = Colour.FromRgb(128, 128, 128)

  Grey #808080
- static Colour SkyBlue = Colour.FromRgb(135, 206, 235)

  SkyBlue #87CEEB
- static Colour LightSkyBlue = Colour.FromRgb(135, 206, 250)

  LightSkyBlue #87CEFA
- static Colour BlueViolet = Colour.FromRgb(138, 43, 226)

  BlueViolet #8A2BE2
- static Colour DarkRed = Colour.FromRgb(139, 0, 0)

  DarkRed #8B0000
- static Colour DarkMagenta = Colour.FromRgb(139, 0, 139)

  DarkMagenta #8B008B
- static Colour SaddleBrown = Colour.FromRgb(139, 69, 19)

  SaddleBrown #8B4513
- static Colour DarkSeaGreen = Colour.FromRgb(143, 188, 143)

  DarkSeaGreen #8FBC8F
- static Colour LightGreen = Colour.FromRgb(144, 238, 144)

  LightGreen #90EE90
- static Colour MediumPurple = Colour.FromRgb(147, 112, 219)

  MediumPurple #9370DB
- static Colour DarkViolet = Colour.FromRgb(148, 0, 211)

  DarkViolet #9400D3
- static Colour PaleGreen = Colour.FromRgb(152, 251, 152)

  PaleGreen #98FB98
- static Colour DarkOrchid = Colour.FromRgb(153, 50, 204)

  DarkOrchid #9932CC

- static Colour YellowGreen = Colour.FromRgb(154, 205, 50)

  YellowGreen #9ACD32
- static Colour Sienna = Colour.FromRgb(160, 82, 45)

  Sienna #A0522D
- static Colour Brown = Colour.FromRgb(165, 42, 42)

  Brown #A52A2A
- static Colour DarkGray = Colour.FromRgb(169, 169, 169)

  DarkGray #A9A9A9
- static Colour DarkGrey = Colour.FromRgb(169, 169, 169)

  DarkGrey #A9A9A9
- static Colour LightBlue = Colour.FromRgb(173, 216, 230)

  LightBlue #ADD8E6
- static Colour GreenYellow = Colour.FromRgb(173, 255, 47)

  GreenYellow #ADFF2F
- static Colour PaleTurquoise = Colour.FromRgb(175, 238, 238)

  PaleTurquoise #AFEEEE
- static Colour LightSteelBlue = Colour.FromRgb(176, 196, 222)

  LightSteelBlue #B0C4DE
- static Colour PowderBlue = Colour.FromRgb(176, 224, 230)

  PowderBlue #B0E0E6
- static Colour FireBrick = Colour.FromRgb(178, 34, 34)

  FireBrick #B22222
- static Colour DarkGoldenRod = Colour.FromRgb(184, 134, 11)

  DarkGoldenRod #B8860B
- static Colour MediumOrchid = Colour.FromRgb(186, 85, 211)

  MediumOrchid #BA55D3
- static Colour RosyBrown = Colour.FromRgb(188, 143, 143)

  RosyBrown #BC8F8F
- static Colour DarkKhaki = Colour.FromRgb(189, 183, 107)

  DarkKhaki #BDB76B
- static Colour Silver = Colour.FromRgb(192, 192, 192)

  Silver #C0C0C0
- static Colour MediumVioletRed = Colour.FromRgb(199, 21, 133)

  MediumVioletRed #C71585
- static Colour IndianRed = Colour.FromRgb(205, 92, 92)

  IndianRed #CD5C5C
- static Colour Peru = Colour.FromRgb(205, 133, 63)

  Peru #CD853F
- static Colour Chocolate = Colour.FromRgb(210, 105, 30)

  Chocolate #D2691E
- static Colour Tan = Colour.FromRgb(210, 180, 140)

  Tan #D2B48C
- static Colour LightGray = Colour.FromRgb(211, 211, 211)

  LightGray #D3D3D3
- static Colour LightGrey = Colour.FromRgb(211, 211, 211)

  LightGrey #D3D3D3
- static Colour Thistle = Colour.FromRgb(216, 191, 216)

  Thistle #D8BFD8
- static Colour Orchid = Colour.FromRgb(218, 112, 214)

  Orchid #DA70D6
- static Colour GoldenRod = Colour.FromRgb(218, 165, 32)

GoldenRod #DAA520

- static Colour PaleVioletRed = Colour.FromRgb(219, 112, 147)

  PaleVioletRed #DB7093
- static Colour Crimson = Colour.FromRgb(220, 20, 60)

  Crimson #DC143C
- static Colour Gainsboro = Colour.FromRgb(220, 220, 220)

  Gainsboro #DCDCDC
- static Colour Plum = Colour.FromRgb(221, 160, 221)

  Plum #DDA0DD
- static Colour BurlyWood = Colour.FromRgb(222, 184, 135)

  BurlyWood #DEB887
- static Colour LightCyan = Colour.FromRgb(224, 255, 255)

  LightCyan #E0FFFF
- static Colour Lavender = Colour.FromRgb(230, 230, 250)

  Lavender #E6E6FA
- static Colour DarkSalmon = Colour.FromRgb(233, 150, 122)

  DarkSalmon #E9967A
- static Colour Violet = Colour.FromRgb(238, 130, 238)

  Violet #EE82EE
- static Colour PaleGoldenRod = Colour.FromRgb(238, 232, 170)

  PaleGoldenRod #EEE8AA
- static Colour LightCoral = Colour.FromRgb(240, 128, 128)

  LightCoral #F08080
- static Colour Khaki = Colour.FromRgb(240, 230, 140)

  Khaki #F0E68C
- static Colour AliceBlue = Colour.FromRgb(240, 248, 255)

  AliceBlue #F0F8FF
- static Colour HoneyDew = Colour.FromRgb(240, 255, 240)

  HoneyDew #F0FFF0
- static Colour Azure = Colour.FromRgb(240, 255, 255)

  Azure #F0FFFF
- static Colour SandyBrown = Colour.FromRgb(244, 164, 96)

  SandyBrown #F4A460
- static Colour Wheat = Colour.FromRgb(245, 222, 179)

  Wheat #F5DEB3
- static Colour Beige = Colour.FromRgb(245, 245, 220)

  Beige #F5F5DC
- static Colour WhiteSmoke = Colour.FromRgb(245, 245, 245)

  WhiteSmoke #F5F5F5
- static Colour MintCream = Colour.FromRgb(245, 255, 250)

  MintCream #F5FFFA
- static Colour GhostWhite = Colour.FromRgb(248, 248, 255)

  GhostWhite #F8F8FF
- static Colour Salmon = Colour.FromRgb(250, 128, 114)

  Salmon #FA8072
- static Colour AntiqueWhite = Colour.FromRgb(250, 235, 215)

  AntiqueWhite #FAEBD7
- static Colour Linen = Colour.FromRgb(250, 240, 230)

  Linen #FAF0E6
- static Colour LightGoldenRodYellow = Colour.FromRgb(250, 250, 210)

  LightGoldenRodYellow #FAFAD2

- static Colour OldLace = Colour.FromRgb(253, 245, 230)

  OldLace #FDF5E6
- static Colour Red = Colour.FromRgb(255, 0, 0)

  Red #FF0000
- static Colour Fuchsia = Colour.FromRgb(255, 0, 255)

  Fuchsia #FF00FF
- static Colour Magenta = Colour.FromRgb(255, 0, 255)

  Magenta #FF00FF
- static Colour DeepPink = Colour.FromRgb(255, 20, 147)

  DeepPink #FF1493
- static Colour OrangeRed = Colour.FromRgb(255, 69, 0)

  OrangeRed #FF4500
- static Colour Tomato = Colour.FromRgb(255, 99, 71)

  Tomato #FF6347
- static Colour HotPink = Colour.FromRgb(255, 105, 180)

  HotPink #FF69B4
- static Colour Coral = Colour.FromRgb(255, 127, 80)

  Coral #FF7F50
- static Colour DarkOrange = Colour.FromRgb(255, 140, 0)

  DarkOrange #FF8C00
- static Colour LightSalmon = Colour.FromRgb(255, 160, 122)

  LightSalmon #FFA07A
- static Colour Orange = Colour.FromRgb(255, 165, 0)

  Orange #FFA500
- static Colour LightPink = Colour.FromRgb(255, 182, 193)

  LightPink #FFB6C1
- static Colour Pink = Colour.FromRgb(255, 192, 203)

  Pink #FFC0CB
- static Colour Gold = Colour.FromRgb(255, 215, 0)

  Gold #FFD700
- static Colour PeachPuff = Colour.FromRgb(255, 218, 185)

  PeachPuff #FFDAB9
- static Colour NavajoWhite = Colour.FromRgb(255, 222, 173)

  NavajoWhite #FFDEAD
- static Colour Moccasin = Colour.FromRgb(255, 228, 181)

  Moccasin #FFE4B5
- static Colour Bisque = Colour.FromRgb(255, 228, 196)

  Bisque #FFE4C4
- static Colour MistyRose = Colour.FromRgb(255, 228, 225)

  MistyRose #FFE4E1
- static Colour BlanchedAlmond = Colour.FromRgb(255, 235, 205)

  BlanchedAlmond #FFEBCD
- static Colour PapayaWhip = Colour.FromRgb(255, 239, 213)

  PapayaWhip #FFEFD5
- static Colour LavenderBlush = Colour.FromRgb(255, 240, 245)

  LavenderBlush #FFF0F5
- static Colour SeaShell = Colour.FromRgb(255, 245, 238)

  SeaShell #FFF5EE
- static Colour Cornsilk = Colour.FromRgb(255, 248, 220)

  Cornsilk #FFF8DC
- static Colour LemonChiffon = Colour.FromRgb(255, 250, 205)

LemonChiffon #FFFACD

- static Colour FloralWhite = Colour.FromRgb(255, 250, 240)

  FloralWhite #FFFAF0
- static Colour Snow = Colour.FromRgb(255, 250, 250)

  Snow #FFFAFA
- static Colour Yellow = Colour.FromRgb(255, 255, 0)

  Yellow #FFFF00
- static Colour LightYellow = Colour.FromRgb(255, 255, 224)

  LightYellow #FFFFE0
- static Colour Ivory = Colour.FromRgb(255, 255, 240)

  Ivory #FFFFF0
- static Colour White = Colour.FromRgb(255, 255, 255)

  White #FFFFFF

7.26.1 Detailed Description

Standard colours.

Definition at line 182 of file StandardColours.cs.

7.26.2 Member Data Documentation

7.26.2.1 AliceBlue

```
Colour VectSharp.Colours.AliceBlue = Colour.FromRgb(240, 248, 255)  [static]
```

AliceBlue #F0F8FF

Definition at line 599 of file StandardColours.cs.

7.26.2.2 AntiqueWhite

```
Colour VectSharp.Colours.AntiqueWhite = Colour.FromRgb(250, 235, 215)  [static]
```

AntiqueWhite #FAEBD7

Definition at line 639 of file StandardColours.cs.

7.26.2.3 Aqua

```
Colour VectSharp.Colours.Aqua = Colour.FromRgb(0, 255, 255)  [static]
```

Aqua #00FFFF

Definition at line 243 of file StandardColours.cs.

7.26.2.4 Aquamarine

```
Colour VectSharp.Colours.Aquamarine = Colour.FromRgb(127, 255, 212)  [static]
```

Aquamarine #7FFFD4

Definition at line 375 of file StandardColours.cs.

7.26.2.5 Azure

```
Colour VectSharp.Colours.Azure = Colour.FromRgb(240, 255, 255)  [static]
```

Azure #F0FFFF

Definition at line 607 of file StandardColours.cs.

7.26.2.6 Beige

```
Colour VectSharp.Colours.Beige = Colour.FromRgb(245, 245, 220)  [static]
```

Beige #F5F5DC

Definition at line 619 of file StandardColours.cs.

7.26.2.7 Bisque

```
Colour VectSharp.Colours.Bisque = Colour.FromRgb(255, 228, 196)  [static]
```

Bisque #FFE4C4

Definition at line 723 of file StandardColours.cs.

7.26.2.8 Black

```
Colour VectSharp.Colours.Black = Colour.FromRgb(0, 0, 0)  [static]
```

Black #000000

Definition at line 187 of file StandardColours.cs.

7.26.2.9 BlanchedAlmond

```
Colour VectSharp.Colours.BlanchedAlmond = Colour.FromRgb(255, 235, 205)  [static]
```

BlanchedAlmond #FFEBCD

Definition at line 731 of file StandardColours.cs.

7.26.2.10 Blue

```
Colour VectSharp.Colours.Blue = Colour.FromRgb(0, 0, 255)  [static]
```

Blue #0000FF

Definition at line 203 of file StandardColours.cs.

7.26.2.11 BlueViolet

```
Colour VectSharp.Colours.BlueViolet = Colour.FromRgb(138, 43, 226)  [static]
```

BlueViolet #8A2BE2

Definition at line 407 of file StandardColours.cs.

7.26.2.12 Brown

```
Colour VectSharp.Colours.Brown = Colour.FromRgb(165, 42, 42)  [static]
```

Brown #A52A2A

Definition at line 455 of file StandardColours.cs.

7.26.2.13 BurlyWood

```
Colour VectSharp.Colours.BurlyWood = Colour.FromRgb(222, 184, 135)  [static]
```

BurlyWood #DEB887

Definition at line 567 of file StandardColours.cs.

7.26.2.14 CadetBlue

```
Colour VectSharp.Colours.CadetBlue = Colour.FromRgb(95, 158, 160)  [static]
```

CadetBlue #5F9EA0

Definition at line 315 of file StandardColours.cs.

7.26.2.15 Chartreuse

```
Colour VectSharp.Colours.Chartreuse = Colour.FromRgb(127, 255, 0)  [static]
```

Chartreuse #7FFF00

Definition at line 371 of file StandardColours.cs.

7.26.2.16 Chocolate

```
Colour VectSharp.Colours.Chocolate = Colour.FromRgb(210, 105, 30)  [static]
```

Chocolate #D2691E

Definition at line 523 of file StandardColours.cs.

7.26.2.17 Coral

```
Colour VectSharp.Colours.Coral = Colour.FromRgb(255, 127, 80)  [static]
```

Coral #FF7F50

Definition at line 683 of file StandardColours.cs.

7.26.2.18 CornflowerBlue

```
Colour VectSharp.Colours.CornflowerBlue = Colour.FromRgb(100, 149, 237)  [static]
```

CornflowerBlue #6495ED

Definition at line 319 of file StandardColours.cs.

7.26.2.19 Cornsilk

```
Colour VectSharp.Colours.Cornsilk = Colour.FromRgb(255, 248, 220)  [static]
```

Cornsilk #FFF8DC

Definition at line 747 of file StandardColours.cs.

7.26.2.20 Crimson

```
Colour VectSharp.Colours.Crimson = Colour.FromRgb(220, 20, 60)  [static]
```

Crimson #DC143C

Definition at line 555 of file StandardColours.cs.

7.26.2.21 Cyan

```
Colour VectSharp.Colours.Cyan = Colour.FromRgb(0, 255, 255)  [static]
```

Cyan #00FFFF

Definition at line 247 of file StandardColours.cs.

7.26.2.22 DarkBlue

```
Colour VectSharp.Colours.DarkBlue = Colour.FromRgb(0, 0, 139)  [static]
```

DarkBlue #00008B

Definition at line 195 of file StandardColours.cs.

7.26.2.23 DarkCyan

```
Colour VectSharp.Colours.DarkCyan = Colour.FromRgb(0, 139, 139)  [static]
```

DarkCyan #008B8B

Definition at line 219 of file StandardColours.cs.

7.26.2.24 DarkGoldenRod

```
Colour VectSharp.Colours.DarkGoldenRod = Colour.FromRgb(184, 134, 11)  [static]
```

DarkGoldenRod #B8860B

Definition at line 491 of file StandardColours.cs.

7.26.2.25 DarkGray

```
Colour VectSharp.Colours.DarkGray = Colour.FromRgb(169, 169, 169)  [static]
```

DarkGray #A9A9A9

Definition at line 459 of file StandardColours.cs.

7.26.2.26 DarkGreen

```
Colour VectSharp.Colours.DarkGreen = Colour.FromRgb(0, 100, 0)  [static]
```

DarkGreen #006400

Definition at line 207 of file StandardColours.cs.

7.26.2.27 DarkGrey

```
Colour VectSharp.Colours.DarkGrey = Colour.FromRgb(169, 169, 169)  [static]
```

DarkGrey #A9A9A9

Definition at line 463 of file StandardColours.cs.

7.26.2.28 DarkKhaki

```
Colour VectSharp.Colours.DarkKhaki = Colour.FromRgb(189, 183, 107)  [static]
```

DarkKhaki #BDB76B

Definition at line 503 of file StandardColours.cs.

7.26.2.29 DarkMagenta

```
Colour VectSharp.Colours.DarkMagenta = Colour.FromRgb(139, 0, 139)  [static]
```

DarkMagenta #8B008B

Definition at line 415 of file StandardColours.cs.

7.26.2.30 DarkOliveGreen

```
Colour VectSharp.Colours.DarkOliveGreen = Colour.FromRgb(85, 107, 47)  [static]
```

DarkOliveGreen #556B2F

Definition at line 311 of file StandardColours.cs.

7.26.2.31 DarkOrange

```
Colour VectSharp.Colours.DarkOrange = Colour.FromRgb(255, 140, 0)  [static]
```

DarkOrange #FF8C00

Definition at line 687 of file StandardColours.cs.

7.26.2.32 DarkOrchid

```
Colour VectSharp.Colours.DarkOrchid = Colour.FromRgb(153, 50, 204)  [static]
```

DarkOrchid #9932CC

Definition at line 443 of file StandardColours.cs.

7.26.2.33 DarkRed

```
Colour VectSharp.Colours.DarkRed = Colour.FromRgb(139, 0, 0)  [static]
```

DarkRed #8B0000

Definition at line 411 of file StandardColours.cs.

7.26.2.34 DarkSalmon

```
Colour VectSharp.Colours.DarkSalmon = Colour.FromRgb(233, 150, 122)  [static]
```

DarkSalmon #E9967A

Definition at line 579 of file StandardColours.cs.

7.26.2.35 DarkSeaGreen

```
Colour VectSharp.Colours.DarkSeaGreen = Colour.FromRgb(143, 188, 143)  [static]
```

DarkSeaGreen #8FBC8F

Definition at line 423 of file StandardColours.cs.

7.26.2.36 DarkSlateBlue

```
Colour VectSharp.Colours.DarkSlateBlue = Colour.FromRgb(72, 61, 139)  [static]
```

DarkSlateBlue #483D8B

Definition at line 299 of file StandardColours.cs.

7.26.2.37 DarkSlateGray

```
Colour VectSharp.Colours.DarkSlateGray = Colour.FromRgb(47, 79, 79)  [static]
```

DarkSlateGray #2F4F4F

Definition at line 271 of file StandardColours.cs.

7.26.2.38 DarkSlateGrey

```
Colour VectSharp.Colours.DarkSlateGrey = Colour.FromRgb(47, 79, 79)  [static]
```

DarkSlateGrey #2F4F4F

Definition at line 275 of file StandardColours.cs.

7.26.2.39 DarkTurquoise

```
Colour VectSharp.Colours.DarkTurquoise = Colour.FromRgb(0, 206, 209)  [static]
```

DarkTurquoise #00CED1

Definition at line 227 of file StandardColours.cs.

7.26.2.40 DarkViolet

```
Colour VectSharp.Colours.DarkViolet = Colour.FromRgb(148, 0, 211)  [static]
```

DarkViolet #9400D3

Definition at line 435 of file StandardColours.cs.

7.26.2.41 DeepPink

```
Colour VectSharp.Colours.DeepPink = Colour.FromRgb(255, 20, 147)  [static]
```

DeepPink #FF1493

Definition at line 667 of file StandardColours.cs.

7.26.2.42 DeepSkyBlue

```
Colour VectSharp.Colours.DeepSkyBlue = Colour.FromRgb(0, 191, 255)  [static]
```

DeepSkyBlue #00BFFF

Definition at line 223 of file StandardColours.cs.

7.26.2.43 DimGray

```
Colour VectSharp.Colours.DimGray = Colour.FromRgb(105, 105, 105)  [static]
```

DimGray #696969

Definition at line 331 of file StandardColours.cs.

7.26.2.44 DimGrey

```
Colour VectSharp.Colours.DimGrey = Colour.FromRgb(105, 105, 105)  [static]
```

DimGrey #696969

Definition at line 335 of file StandardColours.cs.

7.26.2.45 DodgerBlue

```
Colour VectSharp.Colours.DodgerBlue = Colour.FromRgb(30, 144, 255)  [static]
```

DodgerBlue #1E90FF

Definition at line 255 of file StandardColours.cs.

7.26.2.46 FireBrick

```
Colour VectSharp.Colours.FireBrick = Colour.FromRgb(178, 34, 34)  [static]
```

FireBrick #B22222

Definition at line 487 of file StandardColours.cs.

7.26.2.47 FloralWhite

```
Colour VectSharp.Colours.FloralWhite = Colour.FromRgb(255, 250, 240)  [static]
```

FloralWhite #FFFAF0

Definition at line 755 of file StandardColours.cs.

7.26.2.48 ForestGreen

```
Colour VectSharp.Colours.ForestGreen = Colour.FromRgb(34, 139, 34)  [static]
```

ForestGreen #228B22

Definition at line 263 of file StandardColours.cs.

7.26.2.49 Fuchsia

```
Colour VectSharp.Colours.Fuchsia = Colour.FromRgb(255, 0, 255)  [static]
```

Fuchsia #FF00FF

Definition at line 659 of file StandardColours.cs.

7.26.2.50 Gainsboro

```
Colour VectSharp.Colours.Gainsboro = Colour.FromRgb(220, 220, 220)  [static]
```

Gainsboro #DCDCDC

Definition at line 559 of file StandardColours.cs.

7.26.2.51 GhostWhite

```
Colour VectSharp.Colours.GhostWhite = Colour.FromRgb(248, 248, 255)  [static]
```

GhostWhite #F8F8FF

Definition at line 631 of file StandardColours.cs.

7.26.2.52 Gold

```
Colour VectSharp.Colours.Gold = Colour.FromRgb(255, 215, 0)  [static]
```

Gold #FFD700

Definition at line 707 of file StandardColours.cs.

7.26.2.53 GoldenRod

```
Colour VectSharp.Colours.GoldenRod = Colour.FromRgb(218, 165, 32)  [static]
```

GoldenRod #DAA520

Definition at line 547 of file StandardColours.cs.

7.26.2.54 Gray

```
Colour VectSharp.Colours.Gray = Colour.FromRgb(128, 128, 128)  [static]
```

Gray #808080

Definition at line 391 of file StandardColours.cs.

7.26.2.55 Green

```
Colour VectSharp.Colours.Green = Colour.FromRgb(0, 128, 0)  [static]
```

Green #008000

Definition at line 211 of file StandardColours.cs.

7.26.2.56 GreenYellow

```
Colour VectSharp.Colours.GreenYellow = Colour.FromRgb(173, 255, 47)  [static]
```

GreenYellow #ADFF2F

Definition at line 471 of file StandardColours.cs.

7.26.2.57 Grey

```
Colour VectSharp.Colours.Grey = Colour.FromRgb(128, 128, 128)  [static]
```

Grey #808080

Definition at line 395 of file StandardColours.cs.

7.26.2.58 HoneyDew

```
Colour VectSharp.Colours.HoneyDew = Colour.FromRgb(240, 255, 240)  [static]
```

HoneyDew #F0FFF0

Definition at line 603 of file StandardColours.cs.

7.26.2.59 HotPink

```
Colour VectSharp.Colours.HotPink = Colour.FromRgb(255, 105, 180)  [static]
```

HotPink #FF69B4

Definition at line 679 of file StandardColours.cs.

7.26.2.60 IndianRed

```
Colour VectSharp.Colours.IndianRed = Colour.FromRgb(205, 92, 92)  [static]
```

IndianRed #CD5C5C

Definition at line 515 of file StandardColours.cs.

7.26.2.61 Indigo

```
Colour VectSharp.Colours.Indigo = Colour.FromRgb(75, 0, 130)  [static]
```

Indigo #4B0082

Definition at line 307 of file StandardColours.cs.

7.26.2.62 Ivory

```
Colour VectSharp.Colours.Ivory = Colour.FromRgb(255, 255, 240)  [static]
```

Ivory #FFFFF0

Definition at line 771 of file StandardColours.cs.

7.26.2.63 Khaki

```
Colour VectSharp.Colours.Khaki = Colour.FromRgb(240, 230, 140)  [static]
```

Khaki #F0E68C

Definition at line 595 of file StandardColours.cs.

7.26.2.64 Lavender

```
Colour VectSharp.Colours.Lavender = Colour.FromRgb(230, 230, 250)  [static]
```

Lavender #E6E6FA

Definition at line 575 of file StandardColours.cs.

7.26.2.65 LavenderBlush

```
Colour VectSharp.Colours.LavenderBlush = Colour.FromRgb(255, 240, 245)  [static]
```

LavenderBlush #FFF0F5

Definition at line 739 of file StandardColours.cs.

7.26.2.66 LawnGreen

```
Colour VectSharp.Colours.LawnGreen = Colour.FromRgb(124, 252, 0)  [static]
```

LawnGreen #7CFC00

Definition at line 367 of file StandardColours.cs.

7.26.2.67 LemonChiffon

```
Colour VectSharp.Colours.LemonChiffon = Colour.FromRgb(255, 250, 205)  [static]
```

LemonChiffon #FFFACD

Definition at line 751 of file StandardColours.cs.

7.26.2.68 LightBlue

```
Colour VectSharp.Colours.LightBlue = Colour.FromRgb(173, 216, 230)  [static]
```

LightBlue #ADD8E6

Definition at line 467 of file StandardColours.cs.

7.26.2.69 LightCoral

```
Colour VectSharp.Colours.LightCoral = Colour.FromRgb(240, 128, 128)  [static]
```

LightCoral #F08080

Definition at line 591 of file StandardColours.cs.

7.26.2.70 LightCyan

```
Colour VectSharp.Colours.LightCyan = Colour.FromRgb(224, 255, 255)  [static]
```

LightCyan #E0FFFF

Definition at line 571 of file StandardColours.cs.

7.26.2.71 LightGoldenRodYellow

```
Colour VectSharp.Colours.LightGoldenRodYellow = Colour.FromRgb(250, 250, 210)  [static]
```

LightGoldenRodYellow #FAFAD2

Definition at line 647 of file StandardColours.cs.

7.26.2.72 LightGray

```
Colour VectSharp.Colours.LightGray = Colour.FromRgb(211, 211, 211)  [static]
```

LightGray #D3D3D3

Definition at line 531 of file StandardColours.cs.

7.26.2.73 LightGreen

```
Colour VectSharp.Colours.LightGreen = Colour.FromRgb(144, 238, 144)  [static]
```

LightGreen #90EE90

Definition at line 427 of file StandardColours.cs.

7.26.2.74 LightGrey

```
Colour VectSharp.Colours.LightGrey = Colour.FromRgb(211, 211, 211)  [static]
```

LightGrey #D3D3D3

Definition at line 535 of file StandardColours.cs.

7.26.2.75 LightPink

```
Colour VectSharp.Colours.LightPink = Colour.FromRgb(255, 182, 193)  [static]
```

LightPink #FFB6C1

Definition at line 699 of file StandardColours.cs.

7.26.2.76 LightSalmon

```
Colour VectSharp.Colours.LightSalmon = Colour.FromRgb(255, 160, 122)  [static]
```

LightSalmon #FFA07A

Definition at line 691 of file StandardColours.cs.

7.26.2.77 LightSeaGreen

```
Colour VectSharp.Colours.LightSeaGreen = Colour.FromRgb(32, 178, 170)  [static]
```

LightSeaGreen #20B2AA

Definition at line 259 of file StandardColours.cs.

7.26.2.78 LightSkyBlue

```
Colour VectSharp.Colours.LightSkyBlue = Colour.FromRgb(135, 206, 250)  [static]
```

LightSkyBlue #87CEFA

Definition at line 403 of file StandardColours.cs.

7.26.2.79 LightSlateGray

```
Colour VectSharp.Colours.LightSlateGray = Colour.FromRgb(119, 136, 153)  [static]
```

LightSlateGray #778899

Definition at line 355 of file StandardColours.cs.

7.26.2.80 LightSlateGrey

```
Colour VectSharp.Colours.LightSlateGrey = Colour.FromRgb(119, 136, 153)  [static]
```

LightSlateGrey #778899

Definition at line 359 of file StandardColours.cs.

7.26.2.81 LightSteelBlue

```
Colour VectSharp.Colours.LightSteelBlue = Colour.FromRgb(176, 196, 222)  [static]
```

LightSteelBlue #B0C4DE

Definition at line 479 of file StandardColours.cs.

7.26.2.82 LightYellow

```
Colour VectSharp.Colours.LightYellow = Colour.FromRgb(255, 255, 224)  [static]
```

LightYellow #FFFFE0

Definition at line 767 of file StandardColours.cs.

7.26.2.83 Lime

```
Colour VectSharp.Colours.Lime = Colour.FromRgb(0, 255, 0)  [static]
```

Lime #00FF00

Definition at line 235 of file StandardColours.cs.

7.26.2.84 LimeGreen

```
Colour VectSharp.Colours.LimeGreen = Colour.FromRgb(50, 205, 50)  [static]
```

LimeGreen #32CD32

Definition at line 279 of file StandardColours.cs.

7.26.2.85 Linen

```
Colour VectSharp.Colours.Linen = Colour.FromRgb(250, 240, 230)  [static]
```

Linen #FAF0E6

Definition at line 643 of file StandardColours.cs.

7.26.2.86 Magenta

```
Colour VectSharp.Colours.Magenta = Colour.FromRgb(255, 0, 255)  [static]
```

Magenta #FF00FF

Definition at line 663 of file StandardColours.cs.

7.26.2.87 Maroon

```
Colour VectSharp.Colours.Maroon = Colour.FromRgb(128, 0, 0)  [static]
```

Maroon #800000

Definition at line 379 of file StandardColours.cs.

7.26.2.88 MediumAquaMarine

```
Colour VectSharp.Colours.MediumAquaMarine = Colour.FromRgb(102, 205, 170)  [static]
```

MediumAquaMarine #66CDAA

Definition at line 327 of file StandardColours.cs.

7.26.2.89 MediumBlue

```
Colour VectSharp.Colours.MediumBlue = Colour.FromRgb(0, 0, 205)  [static]
```

MediumBlue #0000CD

Definition at line 199 of file StandardColours.cs.

7.26.2.90 MediumOrchid

```
Colour VectSharp.Colours.MediumOrchid = Colour.FromRgb(186, 85, 211)  [static]
```

MediumOrchid #BA55D3

Definition at line 495 of file StandardColours.cs.

7.26.2.91 MediumPurple

```
Colour VectSharp.Colours.MediumPurple = Colour.FromRgb(147, 112, 219)  [static]
```

MediumPurple #9370DB

Definition at line 431 of file StandardColours.cs.

7.26.2.92 MediumSeaGreen

```
Colour VectSharp.Colours.MediumSeaGreen = Colour.FromRgb(60, 179, 113)  [static]
```

MediumSeaGreen #3CB371

Definition at line 283 of file StandardColours.cs.

7.26.2.93 MediumSlateBlue

```
Colour VectSharp.Colours.MediumSlateBlue = Colour.FromRgb(123, 104, 238)  [static]
```

MediumSlateBlue #7B68EE

Definition at line 363 of file StandardColours.cs.

7.26.2.94 MediumSpringGreen

```
Colour VectSharp.Colours.MediumSpringGreen = Colour.FromRgb(0, 250, 154)  [static]
```

MediumSpringGreen #00FA9A

Definition at line 231 of file StandardColours.cs.

7.26.2.95 MediumTurquoise

```
Colour VectSharp.Colours.MediumTurquoise = Colour.FromRgb(72, 209, 204)  [static]
```

MediumTurquoise #48D1CC

Definition at line 303 of file StandardColours.cs.

7.26.2.96 MediumVioletRed

```
Colour VectSharp.Colours.MediumVioletRed = Colour.FromRgb(199, 21, 133)  [static]
```

MediumVioletRed #C71585

Definition at line 511 of file StandardColours.cs.

7.26.2.97 MidnightBlue

```
Colour VectSharp.Colours.MidnightBlue = Colour.FromRgb(25, 25, 112)  [static]
```

MidnightBlue #191970

Definition at line 251 of file StandardColours.cs.

7.26.2.98 MintCream

```
Colour VectSharp.Colours.MintCream = Colour.FromRgb(245, 255, 250)  [static]
```

MintCream #F5FFFA

Definition at line 627 of file StandardColours.cs.

7.26.2.99 MistyRose

```
Colour VectSharp.Colours.MistyRose = Colour.FromRgb(255, 228, 225)  [static]
```

MistyRose #FFE4E1

Definition at line 727 of file StandardColours.cs.

7.26.2.100 Moccasin

```
Colour VectSharp.Colours.Moccasin = Colour.FromRgb(255, 228, 181)  [static]
```

Moccasin #FFE4B5

Definition at line 719 of file StandardColours.cs.

7.26.2.101 NavajoWhite

```
Colour VectSharp.Colours.NavajoWhite = Colour.FromRgb(255, 222, 173)  [static]
```

NavajoWhite #FFDEAD

Definition at line 715 of file StandardColours.cs.

7.26.2.102 Navy

```
Colour VectSharp.Colours.Navy = Colour.FromRgb(0, 0, 128)  [static]
```

Navy #000080

Definition at line 191 of file StandardColours.cs.

7.26.2.103 OldLace

```
Colour VectSharp.Colours.OldLace = Colour.FromRgb(253, 245, 230)  [static]
```

OldLace #FDF5E6

Definition at line 651 of file StandardColours.cs.

7.26.2.104 Olive

```
Colour VectSharp.Colours.Olive = Colour.FromRgb(128, 128, 0)  [static]
```

Olive #808000

Definition at line 387 of file StandardColours.cs.

7.26.2.105 OliveDrab

```
Colour VectSharp.Colours.OliveDrab = Colour.FromRgb(107, 142, 35)  [static]
```

OliveDrab #6B8E23

Definition at line 343 of file StandardColours.cs.

7.26.2.106 Orange

```
Colour VectSharp.Colours.Orange = Colour.FromRgb(255, 165, 0)  [static]
```

Orange #FFA500

Definition at line 695 of file StandardColours.cs.

7.26.2.107 OrangeRed

```
Colour VectSharp.Colours.OrangeRed = Colour.FromRgb(255, 69, 0)  [static]
```

OrangeRed #FF4500

Definition at line 671 of file StandardColours.cs.

7.26.2.108 Orchid

```
Colour VectSharp.Colours.Orchid = Colour.FromRgb(218, 112, 214)  [static]
```

Orchid #DA70D6

Definition at line 543 of file StandardColours.cs.

7.26.2.109 PaleGoldenRod

```
Colour VectSharp.Colours.PaleGoldenRod = Colour.FromRgb(238, 232, 170)  [static]
```

PaleGoldenRod #EEE8AA

Definition at line 587 of file StandardColours.cs.

7.26.2.110 PaleGreen

```
Colour VectSharp.Colours.PaleGreen = Colour.FromRgb(152, 251, 152)  [static]
```

PaleGreen #98FB98

Definition at line 439 of file StandardColours.cs.

7.26.2.111 PaleTurquoise

```
Colour VectSharp.Colours.PaleTurquoise = Colour.FromRgb(175, 238, 238)  [static]
```

PaleTurquoise #AFEEEE

Definition at line 475 of file StandardColours.cs.

7.26.2.112 PaleVioletRed

```
Colour VectSharp.Colours.PaleVioletRed = Colour.FromRgb(219, 112, 147)  [static]
```

PaleVioletRed #DB7093

Definition at line 551 of file StandardColours.cs.

7.26.2.113 PapayaWhip

```
Colour VectSharp.Colours.PapayaWhip = Colour.FromRgb(255, 239, 213)  [static]
```

PapayaWhip #FFEFD5

Definition at line 735 of file StandardColours.cs.

7.26.2.114 PeachPuff

```
Colour VectSharp.Colours.PeachPuff = Colour.FromRgb(255, 218, 185)  [static]
```

PeachPuff #FFDAB9

Definition at line 711 of file StandardColours.cs.

7.26.2.115 Peru

```
Colour VectSharp.Colours.Peru = Colour.FromRgb(205, 133, 63)  [static]
```

Peru #CD853F

Definition at line 519 of file StandardColours.cs.

7.26.2.116 Pink

```
Colour VectSharp.Colours.Pink = Colour.FromRgb(255, 192, 203)  [static]
```

Pink #FFC0CB

Definition at line 703 of file StandardColours.cs.

7.26.2.117 Plum

```
Colour VectSharp.Colours.Plum = Colour.FromRgb(221, 160, 221)  [static]
```

Plum #DDA0DD

Definition at line 563 of file StandardColours.cs.

7.26.2.118 PowderBlue

```
Colour VectSharp.Colours.PowderBlue = Colour.FromRgb(176, 224, 230)  [static]
```

PowderBlue #B0E0E6

Definition at line 483 of file StandardColours.cs.

7.26.2.119 Purple

```
Colour VectSharp.Colours.Purple = Colour.FromRgb(128, 0, 128)  [static]
```

Purple #800080

Definition at line 383 of file StandardColours.cs.

7.26.2.120 RebeccaPurple

```
Colour VectSharp.Colours.RebeccaPurple = Colour.FromRgb(102, 51, 153)  [static]
```

RebeccaPurple #663399

Definition at line 323 of file StandardColours.cs.

7.26.2.121 Red

```
Colour VectSharp.Colours.Red = Colour.FromRgb(255, 0, 0)  [static]
```

Red #FF0000

Definition at line 655 of file StandardColours.cs.

7.26.2.122 RosyBrown

```
Colour VectSharp.Colours.RosyBrown = Colour.FromRgb(188, 143, 143)  [static]
```

RosyBrown #BC8F8F

Definition at line 499 of file StandardColours.cs.

7.26.2.123 RoyalBlue

```
Colour VectSharp.Colours.RoyalBlue = Colour.FromRgb(65, 105, 225)  [static]
```

RoyalBlue #4169E1

Definition at line 291 of file StandardColours.cs.

7.26.2.124 SaddleBrown

```
Colour VectSharp.Colours.SaddleBrown = Colour.FromRgb(139, 69, 19)  [static]
```

SaddleBrown #8B4513

Definition at line 419 of file StandardColours.cs.

7.26.2.125 Salmon

```
Colour VectSharp.Colours.Salmon = Colour.FromRgb(250, 128, 114)  [static]
```

Salmon #FA8072

Definition at line 635 of file StandardColours.cs.

7.26.2.126 SandyBrown

```
Colour VectSharp.Colours.SandyBrown = Colour.FromRgb(244, 164, 96)  [static]
```

SandyBrown #F4A460

Definition at line 611 of file StandardColours.cs.

7.26.2.127 SeaGreen

```
Colour VectSharp.Colours.SeaGreen = Colour.FromRgb(46, 139, 87)  [static]
```

SeaGreen #2E8B57

Definition at line 267 of file StandardColours.cs.

7.26.2.128 SeaShell

```
Colour VectSharp.Colours.SeaShell = Colour.FromRgb(255, 245, 238)  [static]
```

SeaShell #FFF5EE

Definition at line 743 of file StandardColours.cs.

7.26.2.129 Sienna

```
Colour VectSharp.Colours.Sienna = Colour.FromRgb(160, 82, 45)  [static]
```

Sienna #A0522D

Definition at line 451 of file StandardColours.cs.

7.26.2.130 Silver

```
Colour VectSharp.Colours.Silver = Colour.FromRgb(192, 192, 192)  [static]
```

Silver #C0C0C0

Definition at line 507 of file StandardColours.cs.

7.26.2.131 SkyBlue

```
Colour VectSharp.Colours.SkyBlue = Colour.FromRgb(135, 206, 235)  [static]
```

SkyBlue #87CEEB

Definition at line 399 of file StandardColours.cs.

7.26.2.132 SlateBlue

```
Colour VectSharp.Colours.SlateBlue = Colour.FromRgb(106, 90, 205)  [static]
```

SlateBlue #6A5ACD

Definition at line 339 of file StandardColours.cs.

7.26.2.133 SlateGray

```
Colour VectSharp.Colours.SlateGray = Colour.FromRgb(112, 128, 144)  [static]
```

SlateGray #708090

Definition at line 347 of file StandardColours.cs.

7.26.2.134 SlateGrey

```
Colour VectSharp.Colours.SlateGrey = Colour.FromRgb(112, 128, 144)  [static]
```

SlateGrey #708090

Definition at line 351 of file StandardColours.cs.

7.26.2.135 Snow

```
Colour VectSharp.Colours.Snow = Colour.FromRgb(255, 250, 250)  [static]
```

Snow #FFFAFA

Definition at line 759 of file StandardColours.cs.

7.26.2.136 SpringGreen

```
Colour VectSharp.Colours.SpringGreen = Colour.FromRgb(0, 255, 127)  [static]
```

SpringGreen #00FF7F

Definition at line 239 of file StandardColours.cs.

7.26.2.137 SteelBlue

```
Colour VectSharp.Colours.SteelBlue = Colour.FromRgb(70, 130, 180)  [static]
```

SteelBlue #4682B4

Definition at line 295 of file StandardColours.cs.

7.26.2.138 Tan

```
Colour VectSharp.Colours.Tan = Colour.FromRgb(210, 180, 140)  [static]
```

Tan #D2B48C

Definition at line 527 of file StandardColours.cs.

7.26.2.139 Teal

```
Colour VectSharp.Colours.Teal = Colour.FromRgb(0, 128, 128)  [static]
```

Teal #008080

Definition at line 215 of file StandardColours.cs.

7.26.2.140 Thistle

```
Colour VectSharp.Colours.Thistle = Colour.FromRgb(216, 191, 216)  [static]
```

Thistle #D8BFD8

Definition at line 539 of file StandardColours.cs.

7.26.2.141 Tomato

```
Colour VectSharp.Colours.Tomato = Colour.FromRgb(255, 99, 71)  [static]
```

Tomato #FF6347

Definition at line 675 of file StandardColours.cs.

7.26.2.142 Turquoise

```
Colour VectSharp.Colours.Turquoise = Colour.FromRgb(64, 224, 208)  [static]
```

Turquoise #40E0D0

Definition at line 287 of file StandardColours.cs.

7.26.2.143 Violet

```
Colour VectSharp.Colours.Violet = Colour.FromRgb(238, 130, 238)  [static]
```

Violet #EE82EE

Definition at line 583 of file StandardColours.cs.

7.26.2.144 Wheat

```
Colour VectSharp.Colours.Wheat = Colour.FromRgb(245, 222, 179)  [static]
```

Wheat #F5DEB3

Definition at line 615 of file StandardColours.cs.

7.26.2.145 White

```
Colour VectSharp.Colours.White = Colour.FromRgb(255, 255, 255)  [static]
```

White #FFFFFF

Definition at line 775 of file StandardColours.cs.

7.26.2.146 WhiteSmoke

```
Colour VectSharp.Colours.WhiteSmoke = Colour.FromRgb(245, 245, 245)  [static]
```

WhiteSmoke #F5F5F5

Definition at line 623 of file StandardColours.cs.

7.26.2.147 Yellow

```
Colour VectSharp.Colours.Yellow = Colour.FromRgb(255, 255, 0)  [static]
```

Yellow #FFFF00

Definition at line 763 of file StandardColours.cs.

7.26.2.148 YellowGreen

```
Colour VectSharp.Colours.YellowGreen = Colour.FromRgb(154, 205, 50)  [static]
```

YellowGreen #9ACD32

Definition at line 447 of file StandardColours.cs.

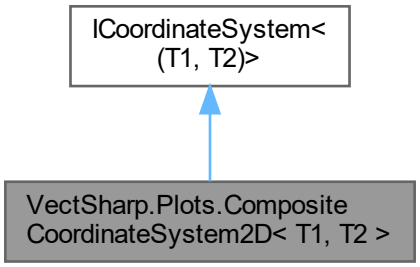
The documentation for this class was generated from the following file:

- VectSharp/StandardColours.cs

7.27 VectSharp.Plots.CompositeCoordinateSystem2D< T1, T2 > Class Template Reference

Combines two ICoordinateSystem1D<T>s to produce a ICoordinateSystem<T>.

Inheritance diagram for VectSharp.Plots.CompositeCoordinateSystem2D< T1, T2 >:



Public Member Functions

- CompositeCoordinateSystem2D (ICoordinateSystem1D< T1 > coordinateSystemX, ICoordinateSystem1D< T2 > coordinateSystemY)

  Creates a new CompositeCoordinateSystem2D<T1, T2> from two 1-D coordinate system.
- Point ToPlotCoordinates ((T1, T2) dataPoint)

Properties

- ICoordinateSystem1D< T1 > CoordinateSystemX [get, set]

  The first coordinate system.
- ICoordinateSystem1D< T2 > CoordinateSystemY [get, set]

  The second coordinate system.

7.27.1 Detailed Description

Combines two ICoordinateSystem1D<T>s to produce a ICoordinateSystem<T>.

Template Parameters

T1	The type for the first coordinate system.
T2	The type for the second coordinate system.

Definition at line 1137 of file CoordinateSystems.cs.

7.27.2 Constructor & Destructor Documentation

7.27.2.1 CompositeCoordinateSystem2D()

```
VectSharp.Plots.CompositeCoordinateSystem2D< T1, T2 >.CompositeCoordinateSystem2D (
            ICoordinateSystem1D< T1 > coordinateSystemX,
            ICoordinateSystem1D< T2 > coordinateSystemY )
```

Creates a new CompositeCoordinateSystem2D<T1, T2> from two 1-D coordinate system.

Parameters

coordinateSystemX	The first coordinate system.
coordinateSystemY	The second coordinate system.

Definition at line 1154 of file CoordinateSystems.cs.

7.27.3 Member Function Documentation

7.27.3.1 ToPlotCoordinates()

```
Point VectSharp.Plots.CompositeCoordinateSystem2D< T1, T2 >.ToPlotCoordinates (
            (T1, T2) dataPoint )
```

Definition at line 1161 of file CoordinateSystems.cs.

7.27.4 Property Documentation

7.27.4.1 CoordinateSystemX

```
ICoordinateSystem1D<T1> VectSharp.Plots.CompositeCoordinateSystem2D< T1, T2 >.Coordinate↩
SystemX [get], [set]
```

The first coordinate system.

Definition at line 1142 of file CoordinateSystems.cs.

7.27.4.2 CoordinateSystemY

```
ICoordinateSystem1D<T2> VectSharp.Plots.CompositeCoordinateSystem2D< T1, T2 >.Coordinate↩
SystemY [get], [set]
```

The second coordinate system.

Definition at line 1147 of file CoordinateSystems.cs.

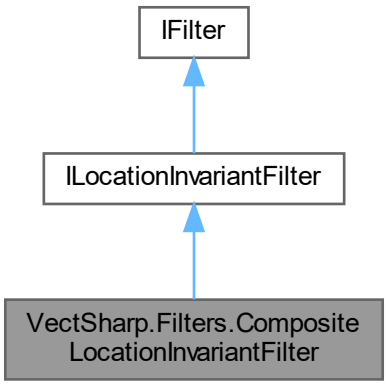
The documentation for this class was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.28 VectSharp.Filters.CompositeLocationInvariantFilter Class Reference

Represents a filter that corresponds to applying multiple ILocationInvariantFilters one after the other.

Inheritance diagram for VectSharp.Filters.CompositeLocationInvariantFilter:



Public Member Functions

- CompositeLocationInvariantFilter (IEnumerable< ILocationInvariantFilter > filters)

  Creates a new CompositeLocationInvariantFilter with the specified filters.
- CompositeLocationInvariantFilter (params ILocationInvariantFilter[ ] filters)

  Creates a new CompositeLocationInvariantFilter with the specified filters.
- RasterImage Filter (RasterImage image, double scale)

  Applies the filter to a RasterImage.

Parameters

image	*The RasterImage to which the filter will be applied.*
scale	*The scale of the image with respect to the filter.*

Returns

A new RasterImage containing the filtered image. The source image is left unaltered.

Properties

- Point TopLeftMargin `[get]`

  Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.
- Point BottomRightMargin `[get]`

  Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.
- ImmutableList< ILocationInvariantFilter > Filters `[get]`

  The filters that are applied by this filter.

7.28.1 Detailed Description

Represents a filter that corresponds to applying multiple ILocationInvariantFilters one after the other.

Definition at line 27 of file CompositeFilter.cs.

7.28.2 Constructor & Destructor Documentation

7.28.2.1 CompositeLocationInvariantFilter() [1/2]

```
VectSharp.Filters.CompositeLocationInvariantFilter.CompositeLocationInvariantFilter (
            IEnumerable< ILocationInvariantFilter > filters )
```

Creates a new CompositeLocationInvariantFilter with the specified filters.

Parameters

filters	The filters that will be applied by the new filter.

Definition at line 44 of file CompositeFilter.cs.

7.28.2.2 CompositeLocationInvariantFilter() [2/2]

```
VectSharp.Filters.CompositeLocationInvariantFilter.CompositeLocationInvariantFilter (
            params ILocationInvariantFilter[] filters )
```

Creates a new CompositeLocationInvariantFilter with the specified filters.

Parameters

filters	The filters that will be applied by the new filter.

Definition at line 71 of file CompositeFilter.cs.

7.28.3 Member Function Documentation

7.28.3.1 Filter()

```
RasterImage VectSharp.Filters.CompositeLocationInvariantFilter.Filter (
            RasterImage image,
            double scale )
```

Applies the filter to a RasterImage.

Parameters

image	The RasterImage to which the filter will be applied.
scale	The scale of the image with respect to the filter.

Returns

A new RasterImage containing the filtered image. The source *image* is left unaltered.

Implements VectSharp.Filters.ILocationInvariantFilter.

Definition at line 93 of file CompositeFilter.cs.

7.28.4 Property Documentation

7.28.4.1 BottomRightMargin

```
Point VectSharp.Filters.CompositeLocationInvariantFilter.BottomRightMargin  [get]
```

Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.

Implements VectSharp.Filters.IFilter.

Definition at line 33 of file CompositeFilter.cs.

7.28.4.2 Filters

```
ImmutableList<ILocationInvariantFilter> VectSharp.Filters.CompositeLocationInvariantFilter.↩
Filters  [get]
```

The filters that are applied by this filter.

Definition at line 38 of file CompositeFilter.cs.

7.28.4.3 TopLeftMargin

```
Point VectSharp.Filters.CompositeLocationInvariantFilter.TopLeftMargin  [get]
```

Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.

Implements VectSharp.Filters.IFilter.

Definition at line 30 of file CompositeFilter.cs.

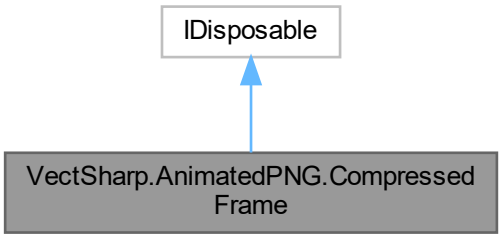
The documentation for this class was generated from the following file:

- VectSharp/Filters/CompositeFilter.cs

7.29 VectSharp.AnimatedPNG.CompressedFrame Class Reference

Represents an individual frame of a PNG animation.

Inheritance diagram for VectSharp.AnimatedPNG.CompressedFrame:



Public Member Functions

- unsafe CompressedFrame (DisposableIntPtr rawFrameData, int width, int height, bool hasAlpha, double duration)

  Creates a new CompressedFrame from raw pixel data in RGB or RGBA format.
- unsafe CompressedFrame (DisposableIntPtr rawFrameData, DisposableIntPtr otherFrameData, bool isFirstFrame, int width, int height, bool hasAlpha, double duration)

  Creates a new CompressedFrame from raw pixel data in RGB or RGBA format, applying inter-frame compression based on another frame.
- void Dispose ()

Properties

- double Duration `[get]`

  The duration of the frame in milliseconds.

7.29.1 Detailed Description

Represents an individual frame of a PNG animation.

Definition at line 56 of file ImageFormats.cs.

7.29.2 Constructor & Destructor Documentation

7.29.2.1 CompressedFrame() `[1/2]`

```
unsafe VectSharp.AnimatedPNG.CompressedFrame.CompressedFrame (
            DisposableIntPtr rawFrameData,
            int width,
            int height,
            bool hasAlpha,
            double duration )
```

Creates a new CompressedFrame from raw pixel data in RGB or RGBA format.

Parameters

rawFrameData	A pointer to the raw pixel data for the frame.
width	The width of the image in pixels.
height	The height of the image in pixels.
hasAlpha	If the image data is in RGBA format, set this to `true`; if it is in RGB format, set it to `false`.
duration	The duration of the frame in milliseconds.

Definition at line 76 of file ImageFormats.cs.

7.29.2.2 CompressedFrame() [2/2]

```
unsafe VectSharp.AnimatedPNG.CompressedFrame.CompressedFrame (
            DisposableIntPtr rawFrameData,
            DisposableIntPtr otherFrameData,
            bool isFirstFrame,
            int width,
            int height,
            bool hasAlpha,
            double duration )
```

Creates a new CompressedFrame from raw pixel data in RGB or RGBA format, applying inter-frame compression based on another frame.

Parameters

rawFrameData	A pointer to the raw pixel data for the new frame.
otherFrameData	A pointer to the raw pixel data for the frame to use as a reference. This can either be the first frame in the animation, or the frame immediately preceeding the current frame.
isFirstFrame	If *otherFrameData* is the raw pixel data for the first frame in the animation, set this to `true`; if it is the immediately preceeding frame, set this to `false`.
width	The width of the image in pixels.
height	The height of the image in pixels.
hasAlpha	If the image data is in RGBA format, set this to `true`; if it is in RGB format, set it to `false`.
duration	The duration of the frame in milliseconds.

Definition at line 93 of file ImageFormats.cs.

7.29.3 Member Function Documentation

7.29.3.1 Dispose()

```
void VectSharp.AnimatedPNG.CompressedFrame.Dispose ( )
```

Definition at line 165 of file ImageFormats.cs.

7.29.4 Property Documentation

7.29.4.1 Duration

```
double VectSharp.AnimatedPNG.CompressedFrame.Duration  [get]
```

The duration of the frame in milliseconds.

Definition at line 64 of file ImageFormats.cs.

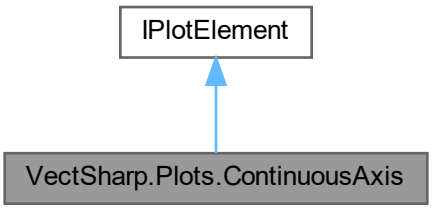
The documentation for this class was generated from the following file:

- VectSharp/ImageFormats.cs

7.30 VectSharp.Plots.ContinuousAxis Class Reference

A plot element that draws an axis line and an arrow.

Inheritance diagram for VectSharp.Plots.ContinuousAxis:



Public Member Functions

- ContinuousAxis (IReadOnlyList< double > startPoint, IReadOnlyList< double > endPoint, IContinuousCoordinateSystem coordinateSystem)

  Creates a new ContinuousAxis instance.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IReadOnlyList< double > StartPoint `[get, set]`

  The starting point of the axis (i.e., the blunt end of the arrow), expressed in data space coordinates.
- IReadOnlyList< double > EndPoint `[get, set]`

  The ending point of the axis (i.e., the pointy end of the arrow), expressed in data space coordinates.
- double ArrowSize = 3 `[get, set]`

  The size of the arrow at the end of the axis, expressed in plot space coordinates.
- IContinuousCoordinateSystem CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() `[get, set]`

  Presentation attributes determining the appearance of the axis.
- string Tag `[get, set]`

  A tag to identify the axis in the plot.

7.30.1 Detailed Description

A plot element that draws an axis line and an arrow.

Definition at line 26 of file Axes.cs.

7.30.2 Constructor & Destructor Documentation

7.30.2.1 ContinuousAxis()

```
VectSharp.Plots.ContinuousAxis.ContinuousAxis (
            IReadOnlyList< double > startPoint,
            IReadOnlyList< double > endPoint,
            IContinuousCoordinateSystem coordinateSystem )
```

Creates a new ContinuousAxis instance.

Parameters

startPoint	The starting point of the axis (i.e., the blunt end of the arrow), expressed in data space coordinates.
endPoint	The ending point of the axis (i.e., the pointy end of the arrow), expressed in data space coordinates.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 65 of file Axes.cs.

7.30.3 Member Function Documentation

7.30.3.1 Plot()

```
void VectSharp.Plots.ContinuousAxis.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 73 of file Axes.cs.

7.30.4 Property Documentation

7.30.4.1 ArrowSize

```
double VectSharp.Plots.ContinuousAxis.ArrowSize = 3  [get], [set]
```

The size of the arrow at the end of the axis, expressed in plot space coordinates.

Definition at line 41 of file Axes.cs.

7.30.4.2 CoordinateSystem

```
IContinuousCoordinateSystem VectSharp.Plots.ContinuousAxis.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 46 of file Axes.cs.

7.30.4.3 EndPoint

```
IReadOnlyList<double> VectSharp.Plots.ContinuousAxis.EndPoint  [get], [set]
```

The ending point of the axis (i.e., the pointy end of the arrow), expressed in data space coordinates.

Definition at line 36 of file Axes.cs.

7.30.4.4 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.ContinuousAxis.PresentationAttributes = new
PlotElementPresentationAttributes()  [get], [set]
```

Presentation attributes determining the appearance of the axis.

Definition at line 52 of file Axes.cs.

7.30.4.5 StartPoint

```
IReadOnlyList<double> VectSharp.Plots.ContinuousAxis.StartPoint  [get], [set]
```

The starting point of the axis (i.e., the blunt end of the arrow), expressed in data space coordinates.

Definition at line 31 of file Axes.cs.

7.30.4.6 Tag

```
string VectSharp.Plots.ContinuousAxis.Tag  [get], [set]
```

A tag to identify the axis in the plot.

Definition at line 57 of file Axes.cs.

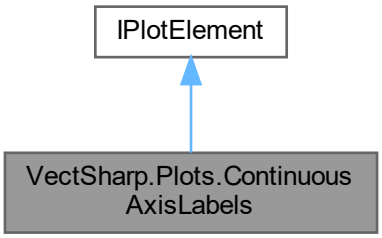
The documentation for this class was generated from the following file:

- VectSharp.Plots/Axes.cs

7.31 VectSharp.Plots.ContinuousAxisLabels Class Reference

A plot element that draws equally spaced labels on an axis.

Inheritance diagram for VectSharp.Plots.ContinuousAxisLabels:



Public Member Functions

- ContinuousAxisLabels (IReadOnlyList< double > startPoint, IReadOnlyList< double > endPoint, IContinuousCoordinateSystem coordinateSystem)

  Create a new ContinuousAxisLabels instance.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IReadOnlyList< double > StartPoint `[get, set]`

  The starting point of the axis, expressed in data space coordinates.
- IReadOnlyList< double > EndPoint `[get, set]`

  The ending point of the axis, expressed in data space coordinates.
- double IntervalCount = 10 `[get, set]`

  The number of intervals between labels. Note that the number of labels will be one greater than this.
- Func< int, double > Position = _ => 10 `[get, set]`

  The distance of the label anchor from the point on the axis. This should be set to a function accepting an `int` argument representing the index of the label, and return a `double` representing the distance of the label from the axis, in plot space coordinates.
- Func< IReadOnlyList< double >, int, IEnumerable< FormattedText > > TextFormat `[get, set]`

  A function used to determine what text to draw at each label. You should set this to a function accepting two parameters: an IReadOnlyList<T> of `double`s, representing the coordinates of the point the label refers to, and an `int` representing the index of the label. The function should return an IEnumerable<T> of FormattedText objects, representing the text that will be drawn.
- double? Rotation = null `[get, set]`

  The orientation of the label with respect to the horizontal. If this is `null`, the labels will be perpendicular to the axis.
- TextBaselines Baseline = TextBaselines.Middle `[get, set]`

  The baseline for the label anchor.
- TextAnchors Alignment = TextAnchors.Left `[get, set]`

  The alignment for the label anchor.
- IContinuousCoordinateSystem CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() `[get, set]`

  Presentation attributes determining the appearance of the labels.
- string Tag `[get, set]`

  A tag to identify the labels in the plot.

7.31.1 Detailed Description

A plot element that draws equally spaced labels on an axis.

Definition at line 309 of file Axes.cs.

7.31.2 Constructor & Destructor Documentation

7.31.2.1 ContinuousAxisLabels()

```
VectSharp.Plots.ContinuousAxisLabels.ContinuousAxisLabels (
            IReadOnlyList< double > startPoint,
            IReadOnlyList< double > endPoint,
            IContinuousCoordinateSystem coordinateSystem )
```

Create a new ContinuousAxisLabels instance.

Parameters

startPoint	The starting point of the axis, expressed in data space coordinates.
endPoint	The ending point of the axis, expressed in data space coordinates.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 454 of file Axes.cs.

7.31.3 Member Function Documentation

7.31.3.1 Plot()

```
void VectSharp.Plots.ContinuousAxisLabels.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 463 of file Axes.cs.

7.31.4 Property Documentation

7.31.4.1 Alignment

```
TextAnchors VectSharp.Plots.ContinuousAxisLabels.Alignment = TextAnchors.Left  [get], [set]
```

The alignment for the label anchor.

Definition at line 430 of file Axes.cs.

7.31.4.2 Baseline

```
TextBaselines VectSharp.Plots.ContinuousAxisLabels.Baseline = TextBaselines.Middle  [get],
[set]
```

The baseline for the label anchor.

Definition at line 425 of file Axes.cs.

7.31.4.3 CoordinateSystem

```
IContinuousCoordinateSystem VectSharp.Plots.ContinuousAxisLabels.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 435 of file Axes.cs.

7.31.4.4 EndPoint

```
IReadOnlyList<double> VectSharp.Plots.ContinuousAxisLabels.EndPoint  [get], [set]
```

The ending point of the axis, expressed in data space coordinates.

Definition at line 319 of file Axes.cs.

7.31.4.5 IntervalCount

```
double VectSharp.Plots.ContinuousAxisLabels.IntervalCount = 10  [get], [set]
```

The number of intervals between labels. Note that the number of labels will be one greater than this.

Definition at line 324 of file Axes.cs.

7.31.4.6 Position

```
Func<int, double> VectSharp.Plots.ContinuousAxisLabels.Position = _ => 10  [get], [set]
```

The distance of the label anchor from the point on the axis. This should be set to a function accepting an `int` argument representing the index of the label, and return a `double` representing the distance of the label from the axis, in plot space coordinates.

Definition at line 331 of file Axes.cs.

7.31.4.7 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.ContinuousAxisLabels.PresentationAttributes
= new PlotElementPresentationAttributes()  [get], [set]
```

Presentation attributes determining the appearance of the labels.

Definition at line 441 of file Axes.cs.

7.31.4.8 Rotation

```
double?  VectSharp.Plots.ContinuousAxisLabels.Rotation = null  [get], [set]
```

The orientation of the label with respect to the horizontal. If this is `null`, the labels will be perpendicular to the axis.

Definition at line 346 of file Axes.cs.

7.31.4.9 StartPoint

```
IReadOnlyList<double> VectSharp.Plots.ContinuousAxisLabels.StartPoint  [get], [set]
```

The starting point of the axis, expressed in data space coordinates.

Definition at line 314 of file Axes.cs.

7.31.4.10 Tag

```
string VectSharp.Plots.ContinuousAxisLabels.Tag  [get], [set]
```

A tag to identify the labels in the plot.

Definition at line 446 of file Axes.cs.

7.31.4.11 TextFormat

```
Func<IReadOnlyList<double>, int, IEnumerable<FormattedText> > VectSharp.Plots.Continuous↩
AxisLabels.TextFormat  [get], [set]
```

A function used to determine what text to draw at each label. You should set this to a function accepting two parameters: an IReadOnlyList<T> of `double`s, representing the coordinates of the point the label refers to, and an `int` representing the index of the label. The function should return an IEnumerable<T> of FormattedText objects, representing the text that will be drawn.

Definition at line 340 of file Axes.cs.

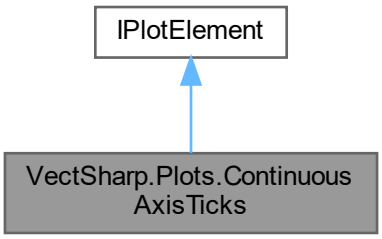
The documentation for this class was generated from the following file:

- VectSharp.Plots/Axes.cs

7.32 VectSharp.Plots.ContinuousAxisTicks Class Reference

A plot element that draws equally spaced ticks on an axis.

Inheritance diagram for VectSharp.Plots.ContinuousAxisTicks:



Public Member Functions

- ContinuousAxisTicks (IReadOnlyList< double > startPoint, IReadOnlyList< double > endPoint, IContinuousCoordinateSystem coordinateSystem)

  Creates a new ContinuousAxisTicks instance.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IReadOnlyList< double > StartPoint `[get, set]`

  The starting point of the axis, expressed in data space coordinates.
- IReadOnlyList< double > EndPoint `[get, set]`

  The ending point of the axis, expressed in data space coordinates.
- double IntervalCount = 10 `[get, set]`

  The number of intervals between ticks. Note that the number of ticks will be one greater than this.
- Func< int, double > SizeAbove = i => i % 2 == 0 ? 3 : 2 `[get, set]`

  The size of the ticks "above" the axis. What "above" means depends on the orientation of the axis. This should be set to a function accepting an `int` argument representing the index of the tick, and return a `double` representing the size of the tick in plot coordinates.
- Func< int, double > SizeBelow = i => i % 2 == 0 ? 3 : 2 `[get, set]`

  The size of the ticks "below" the axis. What "below" means depends on the orientation of the axis. This should be set to a function accepting an `int` argument representing the index of the tick, and return a `double` representing the size of the tick in plot coordinates.
- IContinuousCoordinateSystem CoordinateSystem `[get, set]`

The coordinate system used to transform the points from data space to plot space.

- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() `[get, set]`

  Presentation attributes determining the appearance of the ticks.

- string Tag `[get, set]`

  A tag to identify the ticks in the plot.

7.32.1 Detailed Description

A plot element that draws equally spaced ticks on an axis.

Definition at line 167 of file Axes.cs.

7.32.2 Constructor & Destructor Documentation

7.32.2.1 ContinuousAxisTicks()

```
VectSharp.Plots.ContinuousAxisTicks.ContinuousAxisTicks (
            IReadOnlyList< double > startPoint,
            IReadOnlyList< double > endPoint,
            IContinuousCoordinateSystem coordinateSystem )
```

Creates a new ContinuousAxisTicks instance.

Parameters

startPoint	The starting point of the axis, expressed in data space coordinates.
endPoint	The ending point of the axis, expressed in data space coordinates.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 220 of file Axes.cs.

7.32.3 Member Function Documentation

7.32.3.1 Plot()

```
void VectSharp.Plots.ContinuousAxisTicks.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 228 of file Axes.cs.

7.32.4 Property Documentation

7.32.4.1 CoordinateSystem

```
IContinuousCoordinateSystem VectSharp.Plots.ContinuousAxisTicks.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 201 of file Axes.cs.

7.32.4.2 EndPoint

```
IReadOnlyList<double> VectSharp.Plots.ContinuousAxisTicks.EndPoint  [get], [set]
```

The ending point of the axis, expressed in data space coordinates.

Definition at line 177 of file Axes.cs.

7.32.4.3 IntervalCount

```
double VectSharp.Plots.ContinuousAxisTicks.IntervalCount = 10  [get], [set]
```

The number of intervals between ticks. Note that the number of ticks will be one greater than this.

Definition at line 182 of file Axes.cs.

7.32.4.4 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.ContinuousAxisTicks.PresentationAttributes =
new PlotElementPresentationAttributes()  [get], [set]
```

Presentation attributes determining the appearance of the ticks.

Definition at line 207 of file Axes.cs.

7.32.4.5 SizeAbove

```
Func<int, double> VectSharp.Plots.ContinuousAxisTicks.SizeAbove = i => i % 2 == 0 ?  3 :  2
[get], [set]
```

The size of the ticks "above" the axis. What "above" means depends on the orientation of the axis. This should be set to a function accepting an `int` argument representing the index of the tick, and return a `double` representing the size of the tick in plot coordinates.

Definition at line 189 of file Axes.cs.

7.32.4.6 SizeBelow

```
Func<int, double> VectSharp.Plots.ContinuousAxisTicks.SizeBelow = i => i % 2 == 0 ?  3 :  2
[get], [set]
```

The size of the ticks "below" the axis. What "below" means depends on the orientation of the axis. This should be set to a function accepting an `int` argument representing the index of the tick, and return a `double` representing the size of the tick in plot coordinates.

Definition at line 196 of file Axes.cs.

7.32.4.7 StartPoint

```
IReadOnlyList<double> VectSharp.Plots.ContinuousAxisTicks.StartPoint  [get], [set]
```

The starting point of the axis, expressed in data space coordinates.

Definition at line 172 of file Axes.cs.

7.32.4.8 Tag

```
string VectSharp.Plots.ContinuousAxisTicks.Tag  [get], [set]
```

A tag to identify the ticks in the plot.

Definition at line 212 of file Axes.cs.

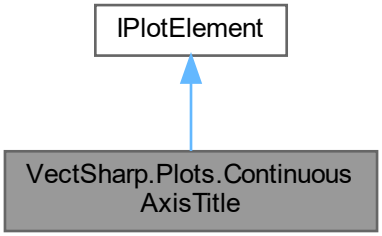
The documentation for this class was generated from the following file:

- VectSharp.Plots/Axes.cs

7.33 VectSharp.Plots.ContinuousAxisTitle Class Reference

A plot element that draws a title for an axis.

Inheritance diagram for VectSharp.Plots.ContinuousAxisTitle:



Public Member Functions

- ContinuousAxisTitle (IEnumerable< FormattedText > title, IReadOnlyList< double > startPoint, IRead↩ OnlyList< double > endPoint, IContinuousCoordinateSystem coordinateSystem)

  Create a new ContinuousAxisTitle instance.
- ContinuousAxisTitle (string title, IReadOnlyList< double > startPoint, IReadOnlyList< double > end↩ Point, IContinuousCoordinateSystem coordinateSystem, PlotElementPresentationAttributes presentation↩ Attributes=null)

  Create a new ContinuousAxisTitle instance.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IReadOnlyList< double > StartPoint [get, set]

  The starting point of the axis, expressed in data space coordinates.
- IReadOnlyList< double > EndPoint [get, set]

  The ending point of the axis, expressed in data space coordinates.
- double Position = 30 [get, set]

  The distance between the title and the axis, in plot space coordinates.
- IEnumerable< FormattedText > Title [get, set]

  The axis title to draw.
- bool FollowAxis = true [get, set]

  If the axis is not a straight line (e.g., because the coordinate system is not linear), if this is `true` the title will follow the shape of the axis. If this is `false`, the title will always be drawn on a straight line.
- double? Rotation = null [get, set]

  Orientation of the title with respect to the horizontal. If this is `null`, the title is parallel to the axis.
- TextBaselines Baseline = TextBaselines.Middle [get, set]

  The baseline for the title anchor.
- TextAnchors Alignment = TextAnchors.Left [get, set]

  The alignment for the title anchor.
- IContinuousCoordinateSystem CoordinateSystem [get, set]

  The coordinate system used to transform the points from data space to plot space.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() { Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 14), Stroke = null, Fill = Colours.Black } [get, set]

  Presentation attributes determining the appearance of the title.
- string Tag [get, set]

  A tag to identify the labels in the plot.

7.33.1 Detailed Description

A plot element that draws a title for an axis.

Definition at line 570 of file Axes.cs.

7.33.2 Constructor & Destructor Documentation

7.33.2.1 ContinuousAxisTitle() [1/2]

```
VectSharp.Plots.ContinuousAxisTitle.ContinuousAxisTitle (
            IEnumerable< FormattedText > title,
            IReadOnlyList< double > startPoint,
            IReadOnlyList< double > endPoint,
            IContinuousCoordinateSystem coordinateSystem )
```

Create a new ContinuousAxisTitle instance.

Parameters

title	The title to draw on the axis.
startPoint	The starting point of the axis, expressed in data space coordinates.
endPoint	The ending point of the axis, expressed in data space coordinates.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 638 of file Axes.cs.

7.33.2.2 ContinuousAxisTitle() [2/2]

```
VectSharp.Plots.ContinuousAxisTitle.ContinuousAxisTitle (
            string title,
            IReadOnlyList< double > startPoint,
            IReadOnlyList< double > endPoint,
            IContinuousCoordinateSystem coordinateSystem,
            PlotElementPresentationAttributes presentationAttributes = null )
```

Create a new ContinuousAxisTitle instance.

Parameters

title	The title to draw on the axis.
startPoint	The starting point of the axis, expressed in data space coordinates.
endPoint	The ending point of the axis, expressed in data space coordinates.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.
presentationAttributes	Presentation attributes determining the appearance of the title.

Definition at line 654 of file Axes.cs.

7.33.3 Member Function Documentation

7.33.3.1 Plot()

```
void VectSharp.Plots.ContinuousAxisTitle.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 685 of file Axes.cs.

7.33.4 Property Documentation

7.33.4.1 Alignment

```
TextAnchors VectSharp.Plots.ContinuousAxisTitle.Alignment = TextAnchors.Left  [get], [set]
```

The alignment for the title anchor.

Definition at line 613 of file Axes.cs.

7.33.4.2 Baseline

```
TextBaselines VectSharp.Plots.ContinuousAxisTitle.Baseline = TextBaselines.Middle  [get], [set]
```

The baseline for the title anchor.

Definition at line 608 of file Axes.cs.

7.33.4.3 CoordinateSystem

```
IContinuousCoordinateSystem VectSharp.Plots.ContinuousAxisTitle.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 618 of file Axes.cs.

7.33.4.4 EndPoint

```
IReadOnlyList<double> VectSharp.Plots.ContinuousAxisTitle.EndPoint  [get], [set]
```

The ending point of the axis, expressed in data space coordinates.

Definition at line 580 of file Axes.cs.

7.33.4.5 FollowAxis

```
bool VectSharp.Plots.ContinuousAxisTitle.FollowAxis = true  [get], [set]
```

If the axis is not a straight line (e.g., because the coordinate system is not linear), if this is `true` the title will follow the shape of the axis. If this is `false`, the title will always be drawn on a straight line.

Definition at line 597 of file Axes.cs.

7.33.4.6 Position

```
double VectSharp.Plots.ContinuousAxisTitle.Position = 30  [get], [set]
```

The distance between the title and the axis, in plot space coordinates.

Definition at line 585 of file Axes.cs.

7.33.4.7 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.ContinuousAxisTitle.PresentationAttributes =
new PlotElementPresentationAttributes() { Font = new Font(FontFamily.ResolveFontFamily(Font↩
Family.StandardFontFamilies.HelveticaBold), 14), Stroke = null, Fill = Colours.Black }  [get],
[set]
```

Presentation attributes determining the appearance of the title.

Definition at line 624 of file Axes.cs.

7.33.4.8 Rotation

```
double?  VectSharp.Plots.ContinuousAxisTitle.Rotation = null  [get], [set]
```

Orientation of the title with respect to the horizontal. If this is `null`, the title is parallel to the axis.

Definition at line 603 of file Axes.cs.

7.33.4.9 StartPoint

```
IReadOnlyList<double> VectSharp.Plots.ContinuousAxisTitle.StartPoint  [get], [set]
```

The starting point of the axis, expressed in data space coordinates.

Definition at line 575 of file Axes.cs.

7.33.4.10 Tag

```
string VectSharp.Plots.ContinuousAxisTitle.Tag  [get], [set]
```

A tag to identify the labels in the plot.

Definition at line 629 of file Axes.cs.

7.33.4.11 Title

```
IEnumerable<FormattedText> VectSharp.Plots.ContinuousAxisTitle.Title  [get], [set]
```

The axis title to draw.

Definition at line 590 of file Axes.cs.

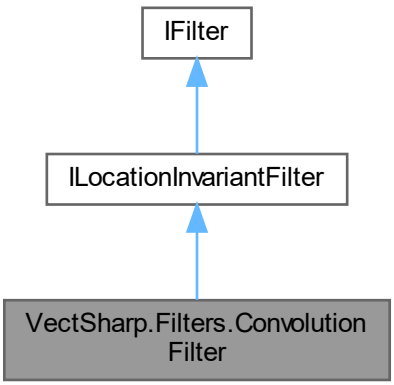
The documentation for this class was generated from the following file:

- VectSharp.Plots/Axes.cs

7.34 VectSharp.Filters.ConvolutionFilter Class Reference

Represents a filter that applies a matrix convolution to the image.

Inheritance diagram for VectSharp.Filters.ConvolutionFilter:



Public Member Functions

- ConvolutionFilter (double[,] kernel, double scale, bool preserveAlpha=true, double normalisation=1, double bias=0)

  Creates a new ConvolutionFilter with the specified parameters.
- virtual RasterImage Filter (RasterImage image, double scale)

  Applies the filter to a RasterImage.

Parameters

image	*The RasterImage to which the filter will be applied.*
scale	*The scale of the image with respect to the filter.*

Returns

A new RasterImage containing the filtered image. The source image is left unaltered.

Properties

- Point TopLeftMargin `[get]`

  Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.
- Point BottomRightMargin `[get]`

  Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.
- virtual double[,] Kernel `[get]`

  The kernel of the ConvolutionFilter. The dimensions of this matrix should all be odd numbers. The larger the kernel, the worse the performance.
- virtual double Normalisation = 1 `[get]`

  The normalisation value that is applies to the kernel.
- virtual double Bias = 0 `[get]`

  The bias value that is added to every colour component when the filter is applied.
- virtual double Scale `[get]`

  The scale relating the size of the kernel to graphics units.
- virtual bool PreserveAlpha = true `[get]`

  If this is `true`, the alpha value of the input pixels is preserved. Otherwise, the alpha channel is subject to the same convolution process as the other colour components.

7.34.1 Detailed Description

Represents a filter that applies a matrix convolution to the image.

Definition at line 27 of file ConvolutionFilter.cs.

7.34.2 Constructor & Destructor Documentation

7.34.2.1 ConvolutionFilter()

```
VectSharp.Filters.ConvolutionFilter.ConvolutionFilter (
            double kernel[,],
            double scale,
            bool preserveAlpha = true,
            double normalisation = 1,
            double bias = 0 )
```

Creates a new ConvolutionFilter with the specified parameters.

Parameters

kernel	The kernel of the ConvolutionFilter. The dimensions of this matrix should all be odd numbers. The larger the kernel, the worse the performance.
scale	The scale relating the size of the kernel to graphics units.
preserveAlpha	If this is `true`, the alpha value of the input pixels is preserved. Otherwise, the alpha channel is subject to the same convolution process as the other colour components.
normalisation	The normalisation value that is applies to the kernel.
bias	The bias value that is added to every colour component when the filter is applied.

Exceptions

ArgumentException	This exception is thrown when the kernel dimensions are not odd numbers.

Definition at line 68 of file ConvolutionFilter.cs.

7.34.3 Member Function Documentation

7.34.3.1 Filter()

```
virtual RasterImage VectSharp.Filters.ConvolutionFilter.Filter (
            RasterImage image,
            double scale )  [virtual]
```

Applies the filter to a RasterImage.

Parameters

image	The RasterImage to which the filter will be applied.
scale	The scale of the image with respect to the filter.

Returns

A new RasterImage containing the filtered image. The source *image* is left unaltered.

Implements VectSharp.Filters.ILocationInvariantFilter.

Definition at line 90 of file ConvolutionFilter.cs.

7.34.4 Property Documentation

7.34.4.1 Bias

```
virtual double VectSharp.Filters.ConvolutionFilter.Bias = 0  [get]
```

The bias value that is added to every colour component when the filter is applied.

Definition at line 47 of file ConvolutionFilter.cs.

7.34.4.2 BottomRightMargin

```
Point VectSharp.Filters.ConvolutionFilter.BottomRightMargin  [get]
```

Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.

Implements VectSharp.Filters.IFilter.

Definition at line 32 of file ConvolutionFilter.cs.

7.34.4.3 Kernel

```
virtual double [,] VectSharp.Filters.ConvolutionFilter.Kernel  [get]
```

The kernel of the ConvolutionFilter. The dimensions of this matrix should all be odd numbers. The larger the kernel, the worse the performance.

Definition at line 37 of file ConvolutionFilter.cs.

7.34.4.4 Normalisation

```
virtual double VectSharp.Filters.ConvolutionFilter.Normalisation = 1  [get]
```

The normalisation value that is applies to the kernel.

Definition at line 42 of file ConvolutionFilter.cs.

7.34.4.5 PreserveAlpha

```
virtual bool VectSharp.Filters.ConvolutionFilter.PreserveAlpha = true  [get]
```

If this is `true`, the alpha value of the input pixels is preserved. Otherwise, the alpha channel is subject to the same convolution process as the other colour components.

Definition at line 57 of file ConvolutionFilter.cs.

7.34.4.6 Scale

```
virtual double VectSharp.Filters.ConvolutionFilter.Scale  [get]
```

The scale relating the size of the kernel to graphics units.

Definition at line 52 of file ConvolutionFilter.cs.

7.34.4.7 TopLeftMargin

```
Point VectSharp.Filters.ConvolutionFilter.TopLeftMargin  [get]
```

Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.

Implements VectSharp.Filters.IFilter.

Definition at line 30 of file ConvolutionFilter.cs.

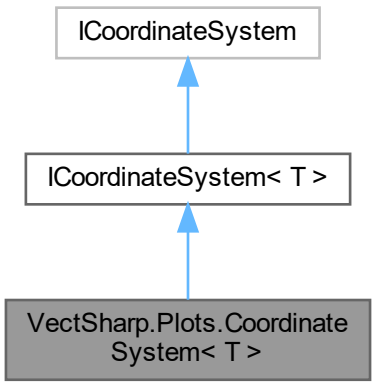
The documentation for this class was generated from the following file:

- VectSharp/Filters/ConvolutionFilter.cs

7.35 VectSharp.Plots.CoordinateSystem< T > Class Template Reference

A coordinate system using a custom method to transform data points.

Inheritance diagram for VectSharp.Plots.CoordinateSystem< T >:



Public Member Functions

- CoordinateSystem (Func< T, Point > coordinateFunction)

  Create a new CoordinateSystem< T > using the specified method.
- Point ToPlotCoordinates (T dataPoint)

  Transform the specified dataPoint into a plot Point.

Parameters

dataPoint	*The data point whose coordinates should be determined.*

Returns

A Point representing the dataPoint in plot space.

Properties

- Func< T, Point > CoordinateFunction `[get, set]`

  The method used to transform the data elements into plot Points.

7.35.1 Detailed Description

A coordinate system using a custom method to transform data points.

Template Parameters

T	The type of data elements.

Definition at line 64 of file CoordinateSystems.cs.

7.35.2 Constructor & Destructor Documentation

7.35.2.1 CoordinateSystem()

```
VectSharp.Plots.CoordinateSystem< T >.CoordinateSystem (
            Func< T, Point > coordinateFunction )
```

Create a new CoordinateSystem<T> using the specified method.

Parameters

coordinateFunction	The method used to transform the data elements into plot Points.

Definition at line 75 of file CoordinateSystems.cs.

7.35.3 Member Function Documentation

7.35.3.1 ToPlotCoordinates()

```
Point VectSharp.Plots.CoordinateSystem< T >.ToPlotCoordinates (
            T dataPoint )
```

Transform the specified *dataPoint* into a plot Point.

Parameters

dataPoint	The data point whose coordinates should be determined.

Returns

A Point representing the *dataPoint* in plot space.

Implements VectSharp.Plots.ICoordinateSystem< T >.

Definition at line 81 of file CoordinateSystems.cs.

7.35.4 Property Documentation

7.35.4.1 CoordinateFunction

```
Func<T, Point> VectSharp.Plots.CoordinateSystem< T >.CoordinateFunction  [get], [set]
```

The method used to transform the data elements into plot Points.

Definition at line 69 of file CoordinateSystems.cs.

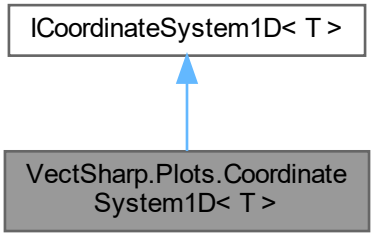
The documentation for this class was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.36 VectSharp.Plots.CoordinateSystem1D< T > Class Template Reference

A coordinate system using a custom method to transform data points.

Inheritance diagram for VectSharp.Plots.CoordinateSystem1D< T >:



Public Member Functions

- CoordinateSystem1D (Func< T, double > coordinateFunction)

  Creates a new CoordinateSystem1D<T> using the specified coordinateFunction .
- double ToPlotCoordinates (T dataPoint)

  Transform the specified dataPoint into a `double`.

Parameters

dataPoint	*The data point whose coordinates should be determined.*

Returns

A `double` *representing the dataPoint in plot space.*

Properties

- Func< T, double > CoordinateFunction `[get, set]`

  The method used to transform data points.

7.36.1 Detailed Description

A coordinate system using a custom method to transform data points.

Template Parameters

T	The type of data elements.

Definition at line 1056 of file CoordinateSystems.cs.

7.36.2 Constructor & Destructor Documentation

7.36.2.1 CoordinateSystem1D()

```
VectSharp.Plots.CoordinateSystem1D< T >.CoordinateSystem1D (
            Func< T, double > coordinateFunction )
```

Creates a new CoordinateSystem1D<T> using the specified *coordinateFunction* .

Parameters

coordinateFunction	The method used to transform data points.

Definition at line 1067 of file CoordinateSystems.cs.

7.36.3 Member Function Documentation

7.36.3.1 ToPlotCoordinates()

```
double VectSharp.Plots.CoordinateSystem1D< T >.ToPlotCoordinates (
            T dataPoint )
```

Transform the specified *dataPoint* into a `double`.

Parameters

dataPoint	The data point whose coordinates should be determined.

Returns

A `double` representing the *dataPoint* in plot space.

Implements VectSharp.Plots.ICoordinateSystem1D< T >.

Definition at line 1073 of file CoordinateSystems.cs.

7.36.4 Property Documentation

7.36.4.1 CoordinateFunction

```
Func<T, double> VectSharp.Plots.CoordinateSystem1D< T >.CoordinateFunction  [get], [set]
```

The method used to transform data points.

Definition at line 1061 of file CoordinateSystems.cs.

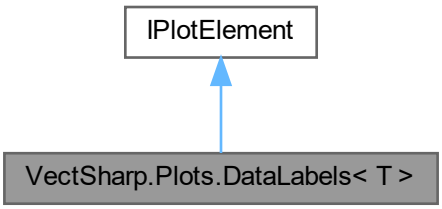
The documentation for this class was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.37 VectSharp.Plots.DataLabels< T > Class Template Reference

A plot element that draws a text label at each data point.

Inheritance diagram for VectSharp.Plots.DataLabels< T >:



Public Member Functions

- DataLabels (IEnumerable< T > data, ICoordinateSystem< T > coordinateSystem)

  Creates a new DataLabels<T> instance.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IEnumerable< T > Data `[get, set]`

  The data points at which the labels will be drawn.
- Func< int, T, object > Label `[get, set]`

  A function used to determine the text of the labels to draw. The arguments for this function should be a `int` representing the index of the data point and a T representing the coordinates of the data point. This function should return either an IEnumerable<T> of FormattedText objects, which will be used as-is, or any other kind of object, which will be converted to a `string` to be used in the label.
- Func< int, T, double > Rotation = (i, d) => 0 `[get, set]`

  A function used to determine the orientation of the labels with respect to the horizontal. The arguments for this function should be a `int` representing the index of the data point and a T representing the coordinates of the data point. This function should return a `double` representing the angle with respect to the horizontal at which the label should be drawn.
- Func< int, T, Point > Margin = (i, d) => new Point() `[get, set]`

  A function used to determine the position of the labels with respect to the data points. The arguments for this function should be a `int` representing the index of the data point and a T representing the coordinates of the data point. This function should return a Point defining the amount of space between the data point and the label, in plot coordinates.
- TextBaselines Baseline = TextBaselines.Middle `[get, set]`

  The baseline for the labels.

- TextAnchors Alignment = TextAnchors.Center [get, set]

  The alignment for the labels.
- ICoordinateSystem< T > CoordinateSystem [get, set]

  The coordinate system used to transform the points from data space to plot space.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() { Stroke = null } [get, set]

  Presentation attributes determining the appearance of the labels.
- string Tag [get, set]

  A tag to identify the labels in the plot.

7.37.1 Detailed Description

A plot element that draws a text label at each data point.

Template Parameters

T	The kind of data describing the data points (generally, `IReadOnlyList<double>`).

Definition at line 336 of file DataPoints.cs.

7.37.2 Constructor & Destructor Documentation

7.37.2.1 DataLabels()

```
VectSharp.Plots.DataLabels< T >.DataLabels (
            IEnumerable< T > data,
            ICoordinateSystem< T > coordinateSystem )
```

Creates a new DataLabels<T> instance.

Parameters

data	The data points at which the labels will be drawn.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 445 of file DataPoints.cs.

7.37.3 Member Function Documentation

7.37.3.1 Plot()

```
void VectSharp.Plots.DataLabels< T >.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 465 of file DataPoints.cs.

7.37.4 Property Documentation

7.37.4.1 Alignment

```
TextAnchors VectSharp.Plots.DataLabels< T >.Alignment = TextAnchors.Center  [get], [set]
```

The alignment for the labels.

Definition at line 422 of file DataPoints.cs.

7.37.4.2 Baseline

```
TextBaselines VectSharp.Plots.DataLabels< T >.Baseline = TextBaselines.Middle  [get], [set]
```

The baseline for the labels.

Definition at line 417 of file DataPoints.cs.

7.37.4.3 CoordinateSystem

```
ICoordinateSystem<T> VectSharp.Plots.DataLabels< T >.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 427 of file DataPoints.cs.

7.37.4.4 Data

```
IEnumerable<T> VectSharp.Plots.DataLabels< T >.Data  [get], [set]
```

The data points at which the labels will be drawn.

Definition at line 341 of file DataPoints.cs.

7.37.4.5 Label

```
Func<int, T, object> VectSharp.Plots.DataLabels< T >.Label  [get], [set]
```

A function used to determine the text of the labels to draw. The arguments for this function should be a `int` representing the index of the data point and a *T* representing the coordinates of the data point. This function should return either an IEnumerable<T> of FormattedText objects, which will be used as-is, or any other kind of object, which will be converted to a `string` to be used in the label.

Definition at line 352 of file DataPoints.cs.

7.37.4.6 Margin

```
Func<int, T, Point> VectSharp.Plots.DataLabels< T >.Margin = (i, d) => new Point()  [get],
[set]
```

A function used to determine the position of the labels with respect to the data points. The arguments for this function should be a `int` representing the index of the data point and a *T* representing the coordinates of the data point. This function should return a Point defining the amount of space between the data point and the label, in plot coordinates.

Definition at line 412 of file DataPoints.cs.

7.37.4.7 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.DataLabels< T >.PresentationAttributes = new
PlotElementPresentationAttributes() { Stroke = null }  [get], [set]
```

Presentation attributes determining the appearance of the labels.

Definition at line 433 of file DataPoints.cs.

7.37.4.8 Rotation

```
Func<int, T, double> VectSharp.Plots.DataLabels< T >.Rotation = (i, d) => 0  [get], [set]
```

A function used to determine the orientation of the labels with respect to the horizontal. The arguments for this function should be a `int` representing the index of the data point and a *T* representing the coordinates of the data point. This function should return a `double` representing the angle with respect to the horizontal at which the label should be drawn.

Definition at line 405 of file DataPoints.cs.

7.37.4.9 Tag

```
string VectSharp.Plots.DataLabels< T >.Tag  [get], [set]
```

A tag to identify the labels in the plot.

Definition at line 438 of file DataPoints.cs.

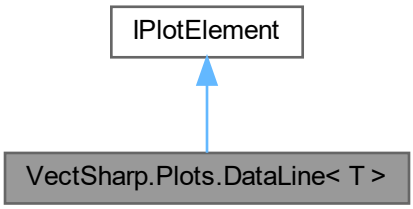
The documentation for this class was generated from the following file:

- VectSharp.Plots/DataPoints.cs

7.38 VectSharp.Plots.DataLine< T > Class Template Reference

A plot element that draws a line passing through a set of points.

Inheritance diagram for VectSharp.Plots.DataLine< T >:



Public Member Functions

- DataLine (IEnumerable< T > data, ICoordinateSystem< T > coordinateSystem)

  Create a new instance of the DataLine< T > class.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IEnumerable< T > Data `[get, set]`

  The data points through which the line will pass.
- bool Smooth `[get, set]`

  If this is `false`, straight line segments are used to join the data points. If this is `true`, a smooth spline passing through all of them is used instead.
- ICoordinateSystem< T > CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() `[get, set]`

  Presentation attributes determining the appearance of the line.
- string Tag `[get, set]`

  A tag to identify the labels in the plot.

7.38.1 Detailed Description

A plot element that draws a line passing through a set of points.

Template Parameters

T	The kind of data describing the data points (generally, `IReadOnlyList<double>`).

Definition at line 515 of file DataPoints.cs.

7.38.2 Constructor & Destructor Documentation

7.38.2.1 DataLine()

```
VectSharp.Plots.DataLine< T >.DataLine (
            IEnumerable< T > data,
            ICoordinateSystem< T > coordinateSystem )
```

Create a new instance of the DataLine<T> class.

Parameters

data	The data points through which the line will pass.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 549 of file DataPoints.cs.

7.38.3 Member Function Documentation

7.38.3.1 Plot()

```
void VectSharp.Plots.DataLine< T >.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 556 of file DataPoints.cs.

7.38.4 Property Documentation

7.38.4.1 CoordinateSystem

```
ICoordinateSystem<T> VectSharp.Plots.DataLine< T >.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 531 of file DataPoints.cs.

7.38.4.2 Data

```
IEnumerable<T> VectSharp.Plots.DataLine< T >.Data  [get], [set]
```

The data points through which the line will pass.

Definition at line 520 of file DataPoints.cs.

7.38.4.3 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.DataLine< T >.PresentationAttributes = new
PlotElementPresentationAttributes()  [get], [set]
```

Presentation attributes determining the appearance of the line.

Definition at line 537 of file DataPoints.cs.

7.38.4.4 Smooth

```
bool VectSharp.Plots.DataLine< T >.Smooth  [get], [set]
```

If this is `false`, straight line segments are used to join the data points. If this is `true`, a smooth spline passing through all of them is used instead.

Definition at line 526 of file DataPoints.cs.

7.38.4.5 Tag

```
string VectSharp.Plots.DataLine< T >.Tag  [get], [set]
```

A tag to identify the labels in the plot.

Definition at line 542 of file DataPoints.cs.

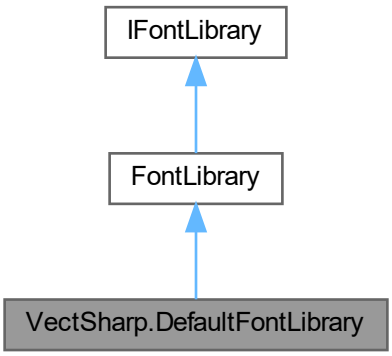
The documentation for this class was generated from the following file:

- VectSharp.Plots/DataPoints.cs

7.39 VectSharp.DefaultFontLibrary Class Reference

A default font library that resolves standard families using the embedded fonts.

Inheritance diagram for VectSharp.DefaultFontLibrary:



Public Member Functions

- override FontFamily ResolveFontFamily (string fontFamily)

  Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised.

Parameters

fontFamily	*The name of the font family to create, or the path to a TTF file.*

Returns

If the font family name or the true type file is valid, a FontFamily object corresponding to the specified font family.

- override FontFamily ResolveFontFamily (FontFamily.StandardFontFamilies standardFontFamily)

  Create a new font family from the specified standard font family name.

Parameters

standardFontFamily	*The standard name of the font family.*

Returns

A FontFamily object corresponding to the specified font family.

7.39.1 Detailed Description

A default font library that resolves standard families using the embedded fonts.

Definition at line 461 of file FontLibrary.cs.

7.39.2 Member Function Documentation

7.39.2.1 ResolveFontFamily() [1/2]

```
override FontFamily VectSharp.DefaultFontLibrary.ResolveFontFamily (
            FontFamily.StandardFontFamilies standardFontFamily )  [virtual]
```

Create a new font family from the specified standard font family name.

Parameters

standardFontFamily	The standard name of the font family.

Returns

A FontFamily object corresponding to the specified font family.

Implements VectSharp.FontLibrary.

Definition at line 523 of file FontLibrary.cs.

7.39.2.2 ResolveFontFamily() [2/2]

```
override FontFamily VectSharp.DefaultFontLibrary.ResolveFontFamily (
            string fontFamily )  [virtual]
```

Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised.

Parameters

fontFamily	The name of the font family to create, or the path to a TTF file.

Returns

If the font family name or the true type file is valid, a FontFamily object corresponding to the specified font family.

Implements VectSharp.FontLibrary.

Definition at line 464 of file FontLibrary.cs.

The documentation for this class was generated from the following file:

- VectSharp/FontLibrary.cs

7.40 VectSharp.Font.DetailedFontMetrics Class Reference

Represents detailed information about the metrics of a text string when drawn with a certain font.

Properties

- double Width [get]

  Width of the text (measured on the actual glyph outlines).
- double Height [get]

  Height of the text (measured on the actual glyph outlines).
- double LeftSideBearing [get]

  How much the leftmost glyph in the string overhangs the glyph origin on the left. Positive for glyphs that hang past the origin (e.g. italic 'f').
- double RightSideBearing [get]

  How much the rightmost glyph in the string overhangs the glyph end on the right. Positive for glyphs that hang past the end (e.g. italic 'f').
- double Top [get]

  Height of the tallest glyph in the string over the baseline. Always >= 0.
- double Bottom [get]

  Depth of the deepest glyph in the string below the baseline. Always <= 0.
- double AdvanceWidth [get]

  Advance width of the text (excluding any left- or right- side bearing).

7.40.1 Detailed Description

Represents detailed information about the metrics of a text string when drawn with a certain font.

Definition at line 102 of file Font.cs.

7.40.2 Property Documentation

7.40.2.1 AdvanceWidth

```
double VectSharp.Font.DetailedFontMetrics.AdvanceWidth  [get]
```

Advance width of the text (excluding any left- or right- side bearing).

Definition at line 137 of file Font.cs.

7.40.2.2 Bottom

```
double VectSharp.Font.DetailedFontMetrics.Bottom  [get]
```

Depth of the deepest glyph in the string below the baseline. Always <= 0.

Definition at line 132 of file Font.cs.

7.40.2.3 Height

```
double VectSharp.Font.DetailedFontMetrics.Height  [get]
```

Height of the text (measured on the actual glyph outlines).

Definition at line 112 of file Font.cs.

7.40.2.4 LeftSideBearing

```
double VectSharp.Font.DetailedFontMetrics.LeftSideBearing  [get]
```

How much the leftmost glyph in the string overhangs the glyph origin on the left. Positive for glyphs that hang past the origin (e.g. italic 'f').

Definition at line 117 of file Font.cs.

7.40.2.5 RightSideBearing

```
double VectSharp.Font.DetailedFontMetrics.RightSideBearing  [get]
```

How much the rightmost glyph in the string overhangs the glyph end on the right. Positive for glyphs that hang past the end (e.g. italic 'f').

Definition at line 122 of file Font.cs.

7.40.2.6 Top

```
double VectSharp.Font.DetailedFontMetrics.Top  [get]
```

Height of the tallest glyph in the string over the baseline. Always >= 0.

Definition at line 127 of file Font.cs.

7.40.2.7 Width

```
double VectSharp.Font.DetailedFontMetrics.Width  [get]
```

Width of the text (measured on the actual glyph outlines).

Definition at line 107 of file Font.cs.

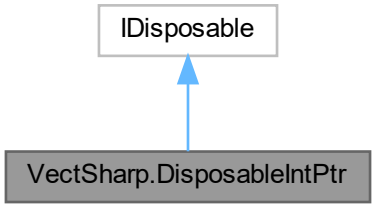
The documentation for this class was generated from the following file:

- VectSharp/Font.cs

7.41 VectSharp.DisposableIntPtr Class Reference

An IDisposable wrapper around an IntPtr that frees the allocated memory when it is disposed.

Inheritance diagram for VectSharp.DisposableIntPtr:



Public Member Functions

- DisposableIntPtr (IntPtr pointer)

  Create a new DisposableIntPtr.
- void Dispose ()

Public Attributes

- readonly IntPtr InternalPointer

  The pointer to the unmanaged memory.

7.41.1 Detailed Description

An IDisposable wrapper around an IntPtr that frees the allocated memory when it is disposed.

Definition at line 54 of file RasterImage.cs.

7.41.2 Constructor & Destructor Documentation

7.41.2.1 DisposableIntPtr()

```
VectSharp.DisposableIntPtr.DisposableIntPtr (
            IntPtr pointer )
```

Create a new DisposableIntPtr.

Parameters

pointer	The pointer that should be freed upon disposing of this object.

Definition at line 65 of file RasterImage.cs.

7.41.3 Member Function Documentation

7.41.3.1 Dispose()

```
void VectSharp.DisposableIntPtr.Dispose ( )
```

Definition at line 89 of file RasterImage.cs.

7.41.4 Member Data Documentation

7.41.4.1 InternalPointer

```
readonly IntPtr VectSharp.DisposableIntPtr.InternalPointer
```

The pointer to the unmanaged memory.

Definition at line 59 of file RasterImage.cs.

The documentation for this class was generated from the following file:

- VectSharp/RasterImage.cs

7.42 VectSharp.Document Class Reference

Represents a collection of pages.

Public Member Functions

- Document ()
  Create a new document.

Public Attributes

- List< Page > Pages = new List<Page>()
  The pages in the document.

7.42.1 Detailed Description

Represents a collection of pages.

Definition at line 27 of file Document.cs.

7.42.2 Constructor & Destructor Documentation

7.42.2.1 Document()

```
VectSharp.Document.Document ( )
```

Create a new document.

Definition at line 38 of file Document.cs.

7.42.3 Member Data Documentation

7.42.3.1 Pages

```
List<Page> VectSharp.Document.Pages = new List<Page>()
```

The pages in the document.

Definition at line 32 of file Document.cs.

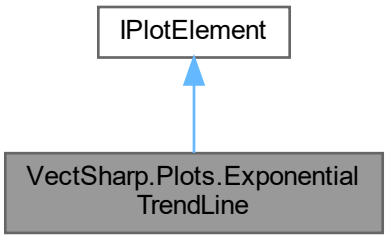
The documentation for this class was generated from the following file:

- VectSharp/Document.cs

7.43 VectSharp.Plots.ExponentialTrendLine Class Reference

A plot element that draws an exponential trendline with equation `y = b * Exp(a * x)`.

Inheritance diagram for VectSharp.Plots.ExponentialTrendLine:



Public Member Functions

- ExponentialTrendLine (double slope, double intercept, double minX, double minY, double maxX, double maxY, IContinuousCoordinateSystem coordinateSystem)

  Create a new ExponentialTrendLine instance, specifying the equation parameters.
- ExponentialTrendLine (IReadOnlyList< IReadOnlyList< double > > data, IContinuousCoordinateSystem coordinateSystem, double? fixedIntercept=null)

  Create a new ExponentialTrendLine instance, determining the equation parameters by running a regression.
- ExponentialTrendLine (IReadOnlyList<(double, double)> data, IContinuousCoordinateSystem coordinate↩System, double? fixedIntercept=null)

  Create a new ExponentialTrendLine instance, determining the equation parameters by running a regression.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- double Slope [get, set]

  The slope of the trendline (a).
- double Intercept [get, set]

  The intercept of the trendline (b).
- double MinX [get, set]

  The minimum X value for which the trendline is plotted.
- double MinY [get, set]

  The minimum Y value for which the trendline is plotted.
- double MaxX [get, set]

  The maximum X value for which the trendline is plotted.
- double MaxY [get, set]

  The maximum Y value for which the trendline is plotted.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180, 180, 180) } [get, set]

  Presentation attributes for the trendline.
- string Tag [get, set]

  A tag to identify the trendline in the plot.
- IContinuousCoordinateSystem CoordinateSystem [get, set]

  The coordinate system used to transform the points from data space to plot space.

7.43.1 Detailed Description

A plot element that draws an exponential trendline with equation `y = b * Exp(a * x)`.

Definition at line 256 of file Trendlines.cs.

7.43.2 Constructor & Destructor Documentation

7.43.2.1 ExponentialTrendLine() [1/3]

```
VectSharp.Plots.ExponentialTrendLine.ExponentialTrendLine (
            double slope,
            double intercept,
            double minX,
            double minY,
            double maxX,
            double maxY,
            IContinuousCoordinateSystem coordinateSystem )
```

Create a new ExponentialTrendLine instance, specifying the equation parameters.

Parameters

slope	The slope of the trendline (a).
intercept	The intercept of the trendline (b).
minX	The minimum X value for which the trendline is plotted.
minY	The minimum Y value for which the trendline is plotted.
maxX	The maximum X value for which the trendline is plotted.
maxY	The maximum Y value for which the trendline is plotted.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 314 of file Trendlines.cs.

7.43.2.2 ExponentialTrendLine() [2/3]

```
VectSharp.Plots.ExponentialTrendLine.ExponentialTrendLine (
            IReadOnlyList< IReadOnlyList< double > > data,
            IContinuousCoordinateSystem coordinateSystem,
            double?  fixedIntercept = null )
```

Create a new ExponentialTrendLine instance, determining the equation parameters by running a regression.

Parameters

data	The data that will be used to determine the equation parameters.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.
fixedIntercept	If this is `null`, the intercept (b) is determined during the regression; otherwise, it is fixed to the specified value.

Definition at line 331 of file Trendlines.cs.

7.43.2.3 ExponentialTrendLine() [3/3]

```
VectSharp.Plots.ExponentialTrendLine.ExponentialTrendLine (
            IReadOnlyList<(double, double)> data,
            IContinuousCoordinateSystem coordinateSystem,
            double?  fixedIntercept = null )
```

Create a new ExponentialTrendLine instance, determining the equation parameters by running a regression.

Parameters

data	The data that will be used to determine the equation parameters.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.
fixedIntercept	If this is `null`, the intercept (b) is determined during the regression; otherwise, it is fixed to the specified value.

Definition at line 385 of file Trendlines.cs.

7.43.3 Member Function Documentation

7.43.3.1 Plot()

```
void VectSharp.Plots.ExponentialTrendLine.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 388 of file Trendlines.cs.

7.43.4 Property Documentation

7.43.4.1 CoordinateSystem

```
IContinuousCoordinateSystem VectSharp.Plots.ExponentialTrendLine.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 301 of file Trendlines.cs.

7.43.4.2 Intercept

```
double VectSharp.Plots.ExponentialTrendLine.Intercept  [get], [set]
```

The intercept of the trendline (b).

Definition at line 266 of file Trendlines.cs.

7.43.4.3 MaxX

```
double VectSharp.Plots.ExponentialTrendLine.MaxX  [get], [set]
```

The maximum X value for which the trendline is plotted.

Definition at line 281 of file Trendlines.cs.

7.43.4.4 MaxY

```
double VectSharp.Plots.ExponentialTrendLine.MaxY  [get], [set]
```

The maximum Y value for which the trendline is plotted.

Definition at line 286 of file Trendlines.cs.

7.43.4.5 MinX

```
double VectSharp.Plots.ExponentialTrendLine.MinX  [get], [set]
```

The minimum X value for which the trendline is plotted.

Definition at line 271 of file Trendlines.cs.

7.43.4.6 MinY

```
double VectSharp.Plots.ExponentialTrendLine.MinY  [get], [set]
```

The minimum Y value for which the trendline is plotted.

Definition at line 276 of file Trendlines.cs.

7.43.4.7 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.ExponentialTrendLine.PresentationAttributes
= new PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180,
180, 180) }  [get], [set]
```

Presentation attributes for the trendline.

Definition at line 291 of file Trendlines.cs.

7.43.4.8 Slope

```
double VectSharp.Plots.ExponentialTrendLine.Slope  [get], [set]
```

The slope of the trendline (a).

Definition at line 261 of file Trendlines.cs.

7.43.4.9 Tag

```
string VectSharp.Plots.ExponentialTrendLine.Tag  [get], [set]
```

A tag to identify the trendline in the plot.

Definition at line 296 of file Trendlines.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/Trendlines.cs

7.44 VectSharp.Canvas.FilterOption Class Reference

Determines how and whether image filters are rasterised.

Public Types

- enum FilterOperations
  Defines whether image filters should be rasterised or not.

Public Member Functions

- FilterOption (FilterOperations operation, double rasterisationResolution, bool rasterisationResolutionRelative)
  Create a new FilterOption object.

Static Public Attributes

- static FilterOption Default = new FilterOption(FilterOperations.RasteriseAllWithSkia, 1, true)
  The default options for image filter rasterisation.

Properties

- FilterOperations Operation = FilterOperations.RasteriseAllWithSkia [get]

  Defines whether image filters should be rasterised or not.
- double RasterisationResolution = 1 [get]

  The resolution that will be used to rasterise image filters. Depending on the value of RasterisationResolutionRelative, this can either be an absolute resolution (i.e. a size in pixel), or a scale factor that is applied to the image size in graphics units.
- bool RasterisationResolutionRelative = true [get]

  Determines whether the value of RasterisationResolution is absolute (i.e. a size in pixel), or relative (i.e. a scale factor that is applied to the image size in graphics units).

7.44.1 Detailed Description

Determines how and whether image filters are rasterised.

Definition at line 1480 of file SKRenderContext.cs.

7.44.2 Member Enumeration Documentation

7.44.2.1 FilterOperations

```
enum VectSharp.Canvas.FilterOption.FilterOperations
```

Defines whether image filters should be rasterised or not.

Definition at line 1485 of file SKRenderContext.cs.

7.44.3 Constructor & Destructor Documentation

7.44.3.1 FilterOption()

```
VectSharp.Canvas.FilterOption.FilterOption (
            FilterOperations operation,
            double rasterisationResolution,
            bool rasterisationResolutionRelative )
```

Create a new FilterOption object.

Parameters

operation	Defines whether image filters should be rasterised or not.
rasterisationResolution	The resolution that will be used to rasterise image filters. Depending on the value of RasterisationResolutionRelative, this can either be an absolute resolution (i.e. a size in pixel), or a scale factor that is applied to the image size in graphics units.
rasterisationResolutionRelative	Determines whether the value of RasterisationResolution is absolute (i.e. a size in pixel), or relative (i.e. a scale factor that is applied to the image size in graphics units).

Definition at line 1534 of file SKRenderContext.cs.

7.44.4 Member Data Documentation

7.44.4.1 Default

```
FilterOption VectSharp.Canvas.FilterOption.Default = new FilterOption(FilterOperations.↩
RasteriseAllWithSkia, 1, true)  [static]
```

The default options for image filter rasterisation.

Definition at line 1526 of file SKRenderContext.cs.

7.44.5 Property Documentation

7.44.5.1 Operation

```
FilterOperations VectSharp.Canvas.FilterOption.Operation = FilterOperations.RasteriseAllWith↩
Skia  [get]
```

Defines whether image filters should be rasterised or not.

Definition at line 1511 of file SKRenderContext.cs.

7.44.5.2 RasterisationResolution

```
double VectSharp.Canvas.FilterOption.RasterisationResolution = 1  [get]
```

The resolution that will be used to rasterise image filters. Depending on the value of RasterisationResolutionRelative, this can either be an absolute resolution (i.e. a size in pixel), or a scale factor that is applied to the image size in graphics units.

Definition at line 1516 of file SKRenderContext.cs.

7.44.5.3 RasterisationResolutionRelative

```
bool VectSharp.Canvas.FilterOption.RasterisationResolutionRelative = true  [get]
```

Determines whether the value of RasterisationResolution is absolute (i.e. a size in pixel), or relative (i.e. a scale factor that is applied to the image size in graphics units).

Definition at line 1521 of file SKRenderContext.cs.

The documentation for this class was generated from the following file:

- VectSharp.Canvas/SKRenderContext.cs

7.45 VectSharp.PDF.PDFContextInterpreter.FilterOption Class Reference

Determines how and whether image filters are rasterised.

Public Types

- enum FilterOperations

  Defines whether image filters should be rasterised or not.

Public Member Functions

- FilterOption (FilterOperations operation, double rasterisationResolution, bool rasterisationResolutionRelative)

  Create a new FilterOption object.

Static Public Attributes

- static FilterOption Default = new FilterOption(FilterOperations.RasteriseAll, 1, true)

  The default options for image filter rasterisation.

Properties

- FilterOperations Operation = FilterOperations.RasteriseAll [get]

  Defines whether image filters should be rasterised or not.
- double RasterisationResolution = 1 [get]

  The resolution that will be used to rasterise image filters. Depending on the value of RasterisationResolutionRelative, this can either be an absolute resolution (i.e. a size in pixel), or a scale factor that is applied to the image size in graphics units.
- bool RasterisationResolutionRelative = true [get]

  Determines whether the value of RasterisationResolution is absolute (i.e. a size in pixel), or relative (i.e. a scale factor that is applied to the image size in graphics units).

7.45.1 Detailed Description

Determines how and whether image filters are rasterised.

Definition at line 107 of file PDFContextInterpreter.cs.

7.45.2 Member Enumeration Documentation

7.45.2.1 FilterOperations

```
enum VectSharp.PDF.PDFContextInterpreter.FilterOption.FilterOperations
```

Defines whether image filters should be rasterised or not.

Definition at line 112 of file PDFContextInterpreter.cs.

7.45.3 Constructor & Destructor Documentation

7.45.3.1 FilterOption()

```
VectSharp.PDF.PDFContextInterpreter.FilterOption.FilterOption (
            FilterOperations operation,
            double rasterisationResolution,
            bool rasterisationResolutionRelative )
```

Create a new FilterOption object.

Parameters

operation	Defines whether image filters should be rasterised or not.
rasterisationResolution	The resolution that will be used to rasterise image filters. Depending on the value of RasterisationResolutionRelative, this can either be an absolute resolution (i.e. a size in pixel), or a scale factor that is applied to the image size in graphics units.
rasterisationResolutionRelative	Determines whether the value of RasterisationResolution is absolute (i.e. a size in pixel), or relative (i.e. a scale factor that is applied to the image size in graphics units).

Definition at line 156 of file PDFContextInterpreter.cs.

7.45.4 Member Data Documentation

7.45.4.1 Default

```
FilterOption VectSharp.PDF.PDFContextInterpreter.FilterOption.Default = new FilterOption(Filter↩
Operations.RasteriseAll, 1, true)  [static]
```

The default options for image filter rasterisation.

Definition at line 148 of file PDFContextInterpreter.cs.

7.45.5 Property Documentation

7.45.5.1 Operation

```
FilterOperations VectSharp.PDF.PDFContextInterpreter.FilterOption.Operation = FilterOperations.↩
RasteriseAll  [get]
```

Defines whether image filters should be rasterised or not.

Definition at line 133 of file PDFContextInterpreter.cs.

7.45.5.2 RasterisationResolution

```
double VectSharp.PDF.PDFContextInterpreter.FilterOption.RasterisationResolution = 1  [get]
```

The resolution that will be used to rasterise image filters. Depending on the value of RasterisationResolutionRelative, this can either be an absolute resolution (i.e. a size in pixel), or a scale factor that is applied to the image size in graphics units.

Definition at line 138 of file PDFContextInterpreter.cs.

7.45.5.3 RasterisationResolutionRelative

```
bool VectSharp.PDF.PDFContextInterpreter.FilterOption.RasterisationResolutionRelative = true
[get]
```

Determines whether the value of RasterisationResolution is absolute (i.e. a size in pixel), or relative (i.e. a scale factor that is applied to the image size in graphics units).

Definition at line 143 of file PDFContextInterpreter.cs.

The documentation for this class was generated from the following file:

- VectSharp.PDF/PDFContextInterpreter.cs

7.46 VectSharp.SVG.SVGContextInterpreter.FilterOption Class Reference

Determines how and whether image filters are rasterised.

Public Types

- enum FilterOperations

  Defines whether image filters should be rasterised or not.

Public Member Functions

- FilterOption (FilterOperations operation, double rasterisationResolution, bool rasterisationResolutionRelative)

  Create a new FilterOption object.

Static Public Attributes

- static FilterOption Default = new FilterOption(FilterOperations.RasteriseIfNecessary, 1, true)

  The default options for image filter rasterisation.

Properties

- FilterOperations Operation = FilterOperations.RasteriseIfNecessary `[get]`

  Defines whether image filters should be rasterised or not.
- double RasterisationResolution = 1 `[get]`

  The resolution that will be used to rasterise image filters. Depending on the value of RasterisationResolutionRelative, this can either be an absolute resolution (i.e. a size in pixel), or a scale factor that is applied to the image size in graphics units.
- bool RasterisationResolutionRelative = true `[get]`

  Determines whether the value of RasterisationResolution is absolute (i.e. a size in pixel), or relative (i.e. a scale factor that is applied to the image size in graphics units).

7.46.1 Detailed Description

Determines how and whether image filters are rasterised.

Definition at line 2378 of file SVGContext.cs.

7.46.2 Member Enumeration Documentation

7.46.2.1 FilterOperations

`enum VectSharp.SVG.SVGContextInterpreter.FilterOption.FilterOperations`

Defines whether image filters should be rasterised or not.

Definition at line 2383 of file SVGContext.cs.

7.46.3 Constructor & Destructor Documentation

7.46.3.1 FilterOption()

```
VectSharp.SVG.SVGContextInterpreter.FilterOption.FilterOption (
            FilterOperations operation,
            double rasterisationResolution,
            bool rasterisationResolutionRelative )
```

Create a new FilterOption object.

Parameters

operation	Defines whether image filters should be rasterised or not.
rasterisationResolution	The resolution that will be used to rasterise image filters. Depending on the value of RasterisationResolutionRelative, this can either be an absolute resolution (i.e. a size in pixel), or a scale factor that is applied to the image size in graphics units.
rasterisationResolutionRelative	Determines whether the value of RasterisationResolution is absolute (i.e. a size in pixel), or relative (i.e. a scale factor that is applied to the image size in graphics units).

Definition at line 2442 of file SVGContext.cs.

7.46.4 Member Data Documentation

7.46.4.1 Default

```
FilterOption VectSharp.SVG.SVGContextInterpreter.FilterOption.Default = new FilterOption(Filter↩
Operations.RasteriseIfNecessary, 1, true)  [static]
```

The default options for image filter rasterisation.

Definition at line 2434 of file SVGContext.cs.

7.46.5 Property Documentation

7.46.5.1 Operation

```
FilterOperations VectSharp.SVG.SVGContextInterpreter.FilterOption.Operation = FilterOperations.↩
RasteriseIfNecessary [get]
```

Defines whether image filters should be rasterised or not.

Definition at line 2419 of file SVGContext.cs.

7.46.5.2 RasterisationResolution

```
double VectSharp.SVG.SVGContextInterpreter.FilterOption.RasterisationResolution = 1 [get]
```

The resolution that will be used to rasterise image filters. Depending on the value of RasterisationResolutionRelative, this can either be an absolute resolution (i.e. a size in pixel), or a scale factor that is applied to the image size in graphics units.

Definition at line 2424 of file SVGContext.cs.

7.46.5.3 RasterisationResolutionRelative

```
bool VectSharp.SVG.SVGContextInterpreter.FilterOption.RasterisationResolutionRelative = true
[get]
```

Determines whether the value of RasterisationResolution is absolute (i.e. a size in pixel), or relative (i.e. a scale factor that is applied to the image size in graphics units).

Definition at line 2429 of file SVGContext.cs.

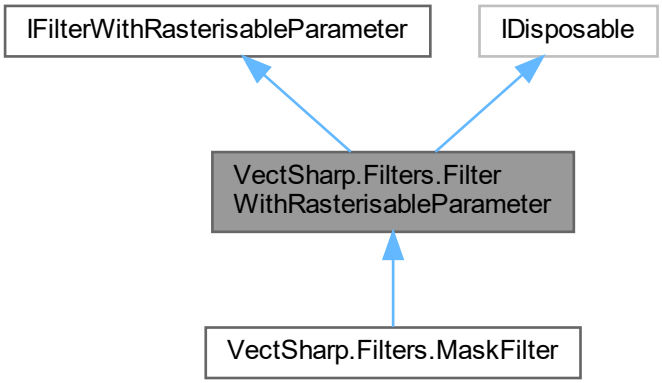
The documentation for this class was generated from the following file:

- VectSharp.SVG/SVGContext.cs

7.47 VectSharp.Filters.FilterWithRasterisableParameter Class Reference

Represents a filter with a parameter that needs to be rasterised at the same resolution as the subject image prior to applying the filter.

Inheritance diagram for VectSharp.Filters.FilterWithRasterisableParameter:



Public Member Functions

- virtual void RasteriseParameter (Func< Graphics, Rectangle, double, bool, RasterImage > rasterisationMethod, double scale)

  Rasterises the filter's parameter at the specified scale, using the specified rasterisation method.

Parameters

rasterisationMethod	*The method used to rasterise the image. The first argument of this method is the Graphics to be rasterised, the second is a Rectangle representing the region to rasterise, the third is a double representing the scale, and the third is a boolean value indicating whether the resulting RasterImage should be interpolated.*
scale	*The scale factor at which the parameter is rasterised.*

- void Dispose ()

7.47.1 Detailed Description

Represents a filter with a parameter that needs to be rasterised at the same resolution as the subject image prior to applying the filter.

Definition at line 89 of file Filters.cs.

7.47.2 Member Function Documentation

7.47.2.1 Dispose()

```
void VectSharp.Filters.FilterWithRasterisableParameter.Dispose ( )
```

Definition at line 190 of file Filters.cs.

7.47.2.2 RasteriseParameter()

```
virtual void VectSharp.Filters.FilterWithRasterisableParameter.RasteriseParameter (
            Func< Graphics, Rectangle, double, bool, RasterImage > rasterisationMethod,
            double scale )  [virtual]
```

Rasterises the filter's parameter at the specified scale, using the specified rasterisation method.

Parameters

rasterisationMethod	The method used to rasterise the image. The first argument of this method is the Graphics to be rasterised, the second is a Rectangle representing the region to rasterise, the third is a double representing the scale, and the third is a boolean value indicating whether the resulting RasterImage should be interpolated.
scale	The scale factor at which the parameter is rasterised.

Implements VectSharp.Filters.IFilterWithRasterisableParameter.

Definition at line 138 of file Filters.cs.

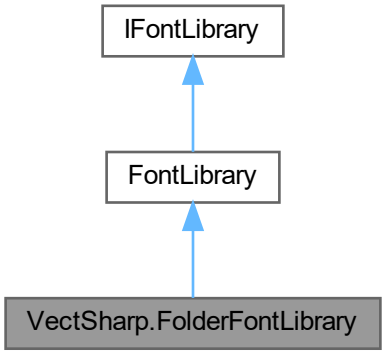
The documentation for this class was generated from the following file:

- VectSharp/Filters/Filters.cs

7.48 VectSharp.FolderFontLibrary Class Reference

A font library that resolves fonts from a folder containing TrueType files.

Inheritance diagram for VectSharp.FolderFontLibrary:



Public Member Functions

- FolderFontLibrary (string folderPath)

  Creates a new FolderFontLibrary using fonts from the specified path, and using the FontFamily.DefaultFontLibrary to resolve the standard font families.
- FolderFontLibrary (string folderPath, IFontLibrary standardFontLibrary)

  Creates a new FolderFontLibrary using fonts from the specified path, and using the specified IFontLibrary to resolve the standard font families.
- override FontFamily ResolveFontFamily (string fontFamily)

  Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised.

Parameters

fontFamily	*The name of the font family to create, or the path to a TTF file.*

Returns

If the font family name or the true type file is valid, a FontFamily object corresponding to the specified font family.

- override FontFamily ResolveFontFamily (FontFamily.StandardFontFamilies standardFontFamily)

  Create a new font family from the specified standard font family name.

Parameters

standardFontFamily	*The standard name of the font family.*

Returns

A FontFamily object corresponding to the specified font family.

7.48.1 Detailed Description

A font library that resolves fonts from a folder containing TrueType files.

Definition at line 555 of file FontLibrary.cs.

7.48.2 Constructor & Destructor Documentation

7.48.2.1 FolderFontLibrary() [1/2]

```
VectSharp.FolderFontLibrary.FolderFontLibrary (
            string folderPath )
```

Creates a new FolderFontLibrary using fonts from the specified path, and using the FontFamily.DefaultFontLibrary to resolve the standard font families.

Parameters

folderPath	The path to the folder containing the TrueType files.

Definition at line 565 of file FontLibrary.cs.

7.48.2.2 FolderFontLibrary() [2/2]

```
VectSharp.FolderFontLibrary.FolderFontLibrary (
            string folderPath,
            IFontLibrary standardFontLibrary )
```

Creates a new FolderFontLibrary using fonts from the specified path, and using the specified IFontLibrary to resolve the standard font families.

Parameters

folderPath	The path to the folder containing the TrueType files.
standardFontLibrary	The IFontLibrary to use when resolving standard font families.

Definition at line 572 of file FontLibrary.cs.

7.48.3 Member Function Documentation

7.48.3.1 ResolveFontFamily() [1/2]

```
override FontFamily VectSharp.FolderFontLibrary.ResolveFontFamily (
            FontFamily.StandardFontFamilies standardFontFamily )  [virtual]
```

Create a new font family from the specified standard font family name.

Parameters

standardFontFamily	The standard name of the font family.

Returns

A FontFamily object corresponding to the specified font family.

Implements VectSharp.FontLibrary.

Definition at line 636 of file FontLibrary.cs.

7.48.3.2 ResolveFontFamily() [2/2]

```
override FontFamily VectSharp.FolderFontLibrary.ResolveFontFamily (
            string fontFamily )  [virtual]
```

Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised.

Parameters

fontFamily	The name of the font family to create, or the path to a TTF file.

Returns

If the font family name or the true type file is valid, a FontFamily object corresponding to the specified font family.

Implements VectSharp.FontLibrary.

Definition at line 604 of file FontLibrary.cs.

The documentation for this class was generated from the following file:

- VectSharp/FontLibrary.cs

7.49 VectSharp.Font Class Reference

Represents a typeface with a specific size.

Classes

- class DetailedFontMetrics

  Represents detailed information about the metrics of a text string when drawn with a certain font.
- class FontUnderline

  Represents options to underline text.

Public Member Functions

- Font (FontFamily fontFamily, double fontSize)

  Create a new Font object, given the base typeface and the font size.
- Font (FontFamily fontFamily, double fontSize, bool underlined)

  Create a new Font object, given the base typeface, the font size, and a boolean value determining whether text using this font should be underlined.
- Font (FontFamily fontFamily, double fontSize, FontUnderline underline)

  Create a new Font object, given the base typeface, the font size, and an object describing the underline properties of text drawn using this font.
- Size MeasureText (string text)

  Measure the size of a text string when typeset with this font.
- DetailedFontMetrics MeasureTextAdvanced (string text)

  Measure all the metrics of a text string when typeset with this font.

Static Public Attributes

- static bool EnableKerning = true

  Determines whether text kerning is enabled. Note that, even when this is set to `false`, text kerning will be applied on some platforms. For the best consistency, leave this set to `true`.

Properties

- double FontSize `[get]`

  Font size, in graphics units.
- FontFamily FontFamily `[get]`

  Font typeface.
- double Ascent `[get]`

  Maximum height over the baseline of the usual glyphs in the font (there may be glyphs taller than this). Always >= 0.
- double WinAscent `[get]`

  Height above the baseline for a clipping region (Windows ascent). Always >= 0.
- double Descent `[get]`

  Maximum depth below the baseline of the usual glyphs in the font (there may be glyphs deeper than this). Always <= 0.
- double YMax `[get]`

  Absolute maximum height over the baseline of the glyphs in the font. Always >= 0.
- double YMin `[get]`

  Absolute maximum depth below the baseline of the glyphs in the font. Always <= 0.
- FontUnderline Underline `[get]`

  Determines the underline style of text drawn using this font. If this is `null`, the text is not underlined.

7.49.1 Detailed Description

Represents a typeface with a specific size.

Definition at line 29 of file Font.cs.

7.49.2 Constructor & Destructor Documentation

7.49.2.1 Font() [1/3]

```
VectSharp.Font.Font (
            FontFamily fontFamily,
            double fontSize )
```

Create a new Font object, given the base typeface and the font size.

Parameters

fontFamily	Base typeface. See FontFamily.
fontSize	The font size, in graphics units.

Definition at line 166 of file Font.cs.

7.49.2.2 Font() [2/3]

```
VectSharp.Font.Font (
            FontFamily fontFamily,
            double fontSize,
            bool underlined )
```

Create a new Font object, given the base typeface, the font size, and a boolean value determining whether text using this font should be underlined.

Parameters

fontFamily	Base typeface. See FontFamily.
fontSize	The font size, in graphics units.
underlined	A boolean value determining whether text drawn using this font should be underlined. The appearance of the underline can be tweaked by changing the properties of the Underline property after the font has been created.

Definition at line 178 of file Font.cs.

7.49.2.3 Font() [3/3]

```
VectSharp.Font.Font (
            FontFamily fontFamily,
            double fontSize,
            FontUnderline underline )
```

Create a new Font object, given the base typeface, the font size, and an object describing the underline properties of text drawn using this font.

Parameters

fontFamily	Base typeface. See FontFamily.
fontSize	The font size, in graphics units.
underline	A FontUnderline object describing the underline properties of text drawn using this font.

Definition at line 195 of file Font.cs.

7.49.3 Member Function Documentation

7.49.3.1 MeasureText()

```
Size VectSharp.Font.MeasureText (
            string text )
```

Measure the size of a text string when typeset with this font.

Parameters

text	The string to measure.

Returns

A Size object representing the width and height of the text.

Definition at line 302 of file Font.cs.

7.49.3.2 MeasureTextAdvanced()

```
DetailedFontMetrics VectSharp.Font.MeasureTextAdvanced (
            string text )
```

Measure all the metrics of a text string when typeset with this font.

Parameters

text	The string to measure.

Returns

A DetailedFontMetrics object representing the metrics of the text.

Definition at line 359 of file Font.cs.

7.49.4 Member Data Documentation

7.49.4.1 EnableKerning

```
bool VectSharp.Font.EnableKerning = true  [static]
```

Determines whether text kerning is enabled. Note that, even when this is set to `false`, text kerning will be applied on some platforms. For the best consistency, leave this set to `true`.

Definition at line 34 of file Font.cs.

7.49.5 Property Documentation

7.49.5.1 Ascent

```
double VectSharp.Font.Ascent  [get]
```

Maximum height over the baseline of the usual glyphs in the font (there may be glyphs taller than this). Always >= 0.

Definition at line 205 of file Font.cs.

7.49.5.2 Descent

```
double VectSharp.Font.Descent  [get]
```

Maximum depth below the baseline of the usual glyphs in the font (there may be glyphs deeper than this). Always <= 0.

Definition at line 241 of file Font.cs.

7.49.5.3 FontFamily

```
FontFamily VectSharp.Font.FontFamily  [get]
```

Font typeface.

Definition at line 159 of file Font.cs.

7.49.5.4 FontSize

```
double VectSharp.Font.FontSize  [get]
```

Font size, in graphics units.

Definition at line 154 of file Font.cs.

7.49.5.5 Underline

```
FontUnderline VectSharp.Font.Underline  [get]
```

Determines the underline style of text drawn using this font. If this is `null`, the text is not underlined.

Definition at line 295 of file Font.cs.

7.49.5.6 WinAscent

```
double VectSharp.Font.WinAscent  [get]
```

Height above the baseline for a clipping region (Windows ascent). Always >= 0.

Definition at line 223 of file Font.cs.

7.49.5.7 YMax

```
double VectSharp.Font.YMax  [get]
```

Absolute maximum height over the baseline of the glyphs in the font. Always >= 0.

Definition at line 259 of file Font.cs.

7.49.5.8 YMin

```
double VectSharp.Font.YMin  [get]
```

Absolute maximum depth below the baseline of the glyphs in the font. Always <= 0.

Definition at line 277 of file Font.cs.

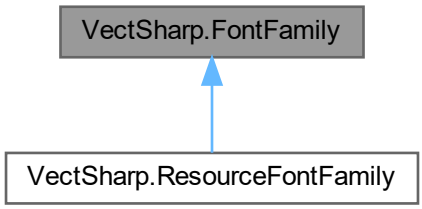
The documentation for this class was generated from the following file:

- VectSharp/Font.cs

7.50 VectSharp.FontFamily Class Reference

Represents a typeface.

Inheritance diagram for VectSharp.FontFamily:



Public Types

- enum StandardFontFamilies

  The 14 standard font families.

Public Member Functions

- FontFamily (string fileName)

  Create a new FontFamily.
- FontFamily (Stream ttfStream)

  Create a new FontFamily.
- FontFamily (TrueTypeFile ttf)

  Create a new FontFamily.
- FontFamily (StandardFontFamilies standardFontFamily)

  Create a new standard FontFamily.

Static Public Member Functions

- static FontFamily ResolveFontFamily (string fontFamily)

  Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised. Equivalent to DefaultFontLibrary.ResolveFontFamily.
- static FontFamily ResolveFontFamily (StandardFontFamilies standardFontFamily)

  Create a new font family from the specified standard font family name. Equivalent to DefaultFontLibrary.ResolveFontFamily.
- static FontFamily ResolveFontFamily (string fontFamily, params string[ ] fallback)

  Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, try to instantiate the font family using the fallback . If none of the fallback family names or true type files are valid, an exception might be raised. Equivalent to DefaultFontLibrary.ResolveFontFamily.
- static FontFamily ResolveFontFamily (string fontFamily, StandardFontFamilies finalFallback, params string[ ] fallback)

  Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, try to instantiate the font family using the fallback . If none of the fallback family names or true type files are valid, instantiate a standard font family using the finalFallback . Equivalent to DefaultFontLibrary.ResolveFontFamily.

Static Public Attributes

- static string[ ] StandardFamilies = new string[ ] { "Times-Roman", "Times-Bold", "Times-Italic", "Times-Bold↩ Italic", "Helvetica", "Helvetica-Bold", "Helvetica-Oblique", "Helvetica-BoldOblique", "Courier", "Courier-Bold", "Courier-Oblique", "Courier-BoldOblique", "Symbol", "ZapfDingbats" }

  The names of the 14 standard families that are guaranteed to be displayed correctly.
- static string[ ] StandardFontFamilyResources

  The names of the resource streams pointing to the included TrueType font files for each of the standard 14 font families.

Properties

- static IFontLibrary DefaultFontLibrary = new DefaultFontLibrary() `[get, set]`

  The default font library used to resolve font family names.
- bool IsStandardFamily `[get]`

  Whether this is one of the 14 standard font families or not.
- string FileName `[get]`

  Full path to the TrueType font file for this font family (or, if this is a standard font family, name of the font family).
- string FamilyName `[get]`

  Name of the font family, including any variantes.
- TrueTypeFile TrueTypeFile `[get]`

  Parsed TrueType font file for this font family. See also:
  See also

  VectSharp.TrueTypeFile

  .
- bool IsBold `[get]`

  Whether this font is bold or not. This is set based on the information included in the OS/2 table of the TrueType file.
- bool IsItalic `[get]`

  Whether this font is italic or oblique or not. This is set based on the information included in the OS/2 table of the TrueType file.
- bool IsOblique `[get]`

  Whether this font is oblique or not. This is set based on the information included in the OS/2 table of the TrueType file.

7.50.1 Detailed Description

Represents a typeface.

Definition at line 423 of file Font.cs.

7.50.2 Member Enumeration Documentation

7.50.2.1 StandardFontFamilies

`enum VectSharp.FontFamily.StandardFontFamilies`

The 14 standard font families.

Definition at line 502 of file Font.cs.

7.50.3 Constructor & Destructor Documentation

7.50.3.1 FontFamily() [1/4]

```
VectSharp.FontFamily.FontFamily (
            string fileName )
```

Create a new FontFamily.

Parameters

fileName	The full path to the TrueType font file for this font family or the name of a standard font family.

Definition at line 611 of file Font.cs.

7.50.3.2 FontFamily() [2/4]

```
VectSharp.FontFamily.FontFamily (
            Stream ttfStream )
```

Create a new FontFamily.

Parameters

ttfStream	A stream containing a file in TTF format.

Definition at line 636 of file Font.cs.

7.50.3.3 FontFamily() [3/4]

```
VectSharp.FontFamily.FontFamily (
            TrueTypeFile ttf )
```

Create a new FontFamily.

Parameters

ttf	A font file in TTF format.

Definition at line 656 of file Font.cs.

7.50.3.4 FontFamily() [4/4]

```
VectSharp.FontFamily.FontFamily (
            StandardFontFamilies standardFontFamily )
```

Create a new standard FontFamily.

Parameters

standardFontFamily	The standard font family.

Definition at line 677 of file Font.cs.

7.50.4 Member Function Documentation

7.50.4.1 ResolveFontFamily() [1/4]

```
static FontFamily VectSharp.FontFamily.ResolveFontFamily (
            StandardFontFamilies standardFontFamily )  [static]
```

Create a new font family from the specified standard font family name. Equivalent to DefaultFontLibrary.ResolveFontFamily.

Parameters

standardFontFamily	The standard name of the font family.

Returns

A FontFamily object corresponding to the specified font family.

7.50.4.2 ResolveFontFamily() [2/4]

```
static FontFamily VectSharp.FontFamily.ResolveFontFamily (
            string fontFamily )  [static]
```

Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised. Equivalent to DefaultFontLibrary.ResolveFontFamily.

Parameters

fontFamily	The name of the font family to create, or the path to a TTF file.

Returns

If the font family name or the true type file is valid, a FontFamily object corresponding to the specified font family.

7.50.4.3 ResolveFontFamily() [3/4]

```
static FontFamily VectSharp.FontFamily.ResolveFontFamily (
            string fontFamily,
            params string[] fallback )  [static]
```

Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, try to instantiate the font family using the *fallback* . If none of the fallback family names or true type files are valid, an exception might be raised. Equivalent to DefaultFontLibrary.ResolveFontFamily.

Parameters

fontFamily	The name of the font family to create, or the path to a TTF file.
fallback	Names of additional font families or TTF files, which will be tried if the first *fontFamily* is not valid.

Returns

A FontFamily object corresponding to the first of the specified font families that is valid.

7.50.4.4 ResolveFontFamily() [4/4]

```
static FontFamily VectSharp.FontFamily.ResolveFontFamily (
            string fontFamily,
            StandardFontFamilies finalFallback,
            params string[] fallback )  [static]
```

Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, try to instantiate the font family using the *fallback* . If none of the fallback family names or true type files are valid, instantiate a standard font family using the *finalFallback* . Equivalent to DefaultFontLibrary.ResolveFontFamily.

Parameters

fontFamily	The name of the font family to create, or the path to a TTF file.
fallback	Names of additional font families or TTF files, which will be tried if the first *fontFamily* is not valid.
finalFallback	The standard name of the font family that will be used if none of the fallback families are valid.

Returns

A FontFamily object corresponding to the first of the specified font families that is valid.

7.50.5 Member Data Documentation

7.50.5.1 StandardFamilies

```
string [] VectSharp.FontFamily.StandardFamilies = new string[] { "Times-Roman", "Times-Bold",
"Times-Italic", "Times-BoldItalic", "Helvetica", "Helvetica-Bold", "Helvetica-Oblique", "Helvetica-Bold↩
Oblique", "Courier", "Courier-Bold", "Courier-Oblique", "Courier-BoldOblique", "Symbol", "Zapf↩
Dingbats" }  [static]
```

The names of the 14 standard families that are guaranteed to be displayed correctly.

Definition at line 481 of file Font.cs.

7.50.5.2 StandardFontFamilyResources

```
string [] VectSharp.FontFamily.StandardFontFamilyResources  [static]
```

Initial value:

```
= new string[]
        {
            "VectSharp.StandardFonts.Tinos-Regular.ttf", "VectSharp.StandardFonts.Tinos-Bold.ttf",
      "VectSharp.StandardFonts.Tinos-Italic.ttf", "VectSharp.StandardFonts.Tinos-BoldItalic.ttf",
            "VectSharp.StandardFonts.Arimo-Regular.ttf", "VectSharp.StandardFonts.Arimo-Bold.ttf",
      "VectSharp.StandardFonts.Arimo-Italic.ttf", "VectSharp.StandardFonts.Arimo-BoldItalic.ttf",
            "VectSharp.StandardFonts.Cousine-Regular.ttf", "VectSharp.StandardFonts.Cousine-Bold.ttf",
      "VectSharp.StandardFonts.Cousine-Italic.ttf", "VectSharp.StandardFonts.Cousine-BoldItalic.ttf",
            "VectSharp.StandardFonts.SymbolNeu_GB.ttf", "VectSharp.StandardFonts.Levibats-Regular_GB.ttf"
        }
```

The names of the resource streams pointing to the included TrueType font files for each of the standard 14 font families.

Definition at line 486 of file Font.cs.

7.50.6 Property Documentation

7.50.6.1 DefaultFontLibrary

```
IFontLibrary VectSharp.FontFamily.DefaultFontLibrary = new DefaultFontLibrary()  [static],
[get], [set]
```

The default font library used to resolve font family names.

Definition at line 428 of file Font.cs.

7.50.6.2 FamilyName

```
string VectSharp.FontFamily.FamilyName  [get]
```

Name of the font family, including any variantes.

Definition at line 583 of file Font.cs.

7.50.6.3 FileName

```
string VectSharp.FontFamily.FileName  [get]
```

Full path to the TrueType font file for this font family (or, if this is a standard font family, name of the font family).

Definition at line 578 of file Font.cs.

7.50.6.4 IsBold

```
bool VectSharp.FontFamily.IsBold  [get]
```

Whether this font is bold or not. This is set based on the information included in the OS/2 table of the TrueType file.

Definition at line 594 of file Font.cs.

7.50.6.5 IsItalic

```
bool VectSharp.FontFamily.IsItalic  [get]
```

Whether this font is italic or oblique or not. This is set based on the information included in the OS/2 table of the TrueType file.

Definition at line 599 of file Font.cs.

7.50.6.6 IsOblique

```
bool VectSharp.FontFamily.IsOblique  [get]
```

Whether this font is oblique or not. This is set based on the information included in the OS/2 table of the TrueType file.

Definition at line 604 of file Font.cs.

7.50.6.7 IsStandardFamily

```
bool VectSharp.FontFamily.IsStandardFamily  [get]
```

Whether this is one of the 14 standard font families or not.

Definition at line 497 of file Font.cs.

7.50.6.8 TrueTypeFile

```
TrueTypeFile VectSharp.FontFamily.TrueTypeFile  [get]
```

Parsed TrueType font file for this font family. See also:

See also

VectSharp.TrueTypeFile

.

Definition at line 589 of file Font.cs.

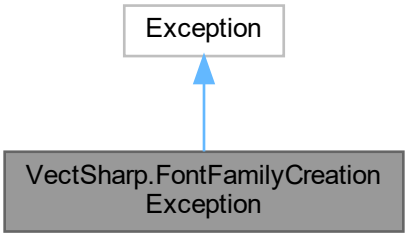
The documentation for this class was generated from the following file:

- VectSharp/Font.cs

7.51 VectSharp.FontFamilyCreationException Class Reference

An exception that occurs while creating a FontFamily.

Inheritance diagram for VectSharp.FontFamilyCreationException:



Public Member Functions

- FontFamilyCreationException (string fontFamily)

  Create a new FontFamilyCreationException instance.

Properties

- string FontFamily `[get]`

  The name of the font family that was being created.

7.51.1 Detailed Description

An exception that occurs while creating a FontFamily.

Definition at line 441 of file FontLibrary.cs.

7.51.2 Constructor & Destructor Documentation

7.51.2.1 FontFamilyCreationException()

```
VectSharp.FontFamilyCreationException.FontFamilyCreationException (
            string fontFamily )
```

Create a new FontFamilyCreationException instance.

Parameters

fontFamily	The name of the font family that was being created.

Definition at line 452 of file FontLibrary.cs.

7.51.3 Property Documentation

7.51.3.1 FontFamily

```
string VectSharp.FontFamilyCreationException.FontFamily [get]
```

The name of the font family that was being created.

Definition at line 446 of file FontLibrary.cs.

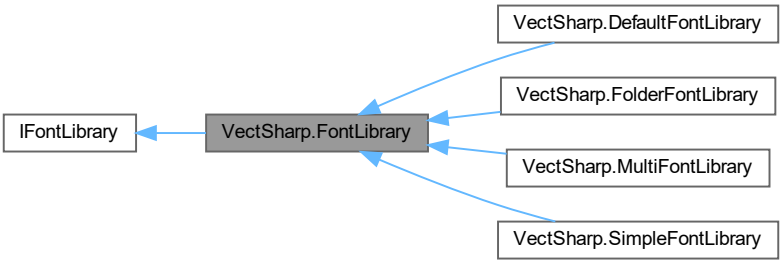
The documentation for this class was generated from the following file:

- VectSharp/FontLibrary.cs

7.52 VectSharp.FontLibrary Class Reference

Abstract class with a default implementation of font family fallbacks.

Inheritance diagram for VectSharp.FontLibrary:



Public Member Functions

- abstract FontFamily ResolveFontFamily (string fontFamily)

  Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised.

Parameters

fontFamily	*The name of the font family to create, or the path to a TTF file.*

Returns

If the font family name or the true type file is valid, a FontFamily object corresponding to the specified font family.

- abstract FontFamily ResolveFontFamily (FontFamily.StandardFontFamilies standardFontFamily)

  Create a new font family from the specified standard font family name.

Parameters

standardFontFamily	*The standard name of the font family.*

Returns

A FontFamily object corresponding to the specified font family.

- virtual FontFamily ResolveFontFamily (string fontFamily, params string[ ] fallback)

  Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, try to instantiate the font family using the fallback . If none of the fallback family names or true type files are valid, an exception might be raised.

Parameters

fontFamily	*The name of the font family to create, or the path to a TTF file.*
fallback	*Names of additional font families or TTF files, which will be tried if the first fontFamily is not valid.*

Returns

A FontFamily object corresponding to the first of the specified font families that is valid.

- virtual FontFamily ResolveFontFamily (string fontFamily, FontFamily.StandardFontFamilies finalFallback, params string[ ] fallback)

  Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, try to instantiate the font family using the fallback . If none of the fallback family names or true type files are valid, instantiate a standard font family using the finalFallback .

Parameters

fontFamily	*The name of the font family to create, or the path to a TTF file.*
fallback	*Names of additional font families or TTF files, which will be tried if the first fontFamily is not valid.*
finalFallback	*The standard name of the font family that will be used if none of the fallback families are valid.*

Returns

A FontFamily object corresponding to the first of the specified font families that is valid.

7.52.1 Detailed Description

Abstract class with a default implementation of font family fallbacks.

Definition at line 67 of file FontLibrary.cs.

7.52.2 Member Function Documentation

7.52.2.1 ResolveFontFamily() [1/4]

```
abstract FontFamily VectSharp.FontLibrary.ResolveFontFamily (
            FontFamily.StandardFontFamilies standardFontFamily )  [pure virtual]
```

Create a new font family from the specified standard font family name.

Parameters

standardFontFamily	The standard name of the font family.

Returns

A FontFamily object corresponding to the specified font family.

Implements VectSharp.IFontLibrary.

Implemented in VectSharp.SimpleFontLibrary, VectSharp.DefaultFontLibrary, VectSharp.FolderFontLibrary, and VectSharp.MultiFontLibrary.

7.52.2.2 ResolveFontFamily() [2/4]

```
abstract FontFamily VectSharp.FontLibrary.ResolveFontFamily (
            string fontFamily )  [pure virtual]
```

Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised.

Parameters

fontFamily	The name of the font family to create, or the path to a TTF file.

Returns

If the font family name or the true type file is valid, a FontFamily object corresponding to the specified font family.

Implements VectSharp.IFontLibrary.

Implemented in VectSharp.SimpleFontLibrary, VectSharp.DefaultFontLibrary, VectSharp.FolderFontLibrary, and VectSharp.MultiFontLibrary.

7.52.2.3 ResolveFontFamily() [3/4]

```
virtual FontFamily VectSharp.FontLibrary.ResolveFontFamily (
            string fontFamily,
            FontFamily.StandardFontFamilies finalFallback,
            params string[] fallback )  [virtual]
```

Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, try to instantiate the font family using the *fallback* . If none of the fallback family names or true type files are valid, instantiate a standard font family using the *finalFallback* .

Parameters

fontFamily	The name of the font family to create, or the path to a TTF file.
fallback	Names of additional font families or TTF files, which will be tried if the first *fontFamily* is not valid.
finalFallback	The standard name of the font family that will be used if none of the fallback families are valid.

Returns

A FontFamily object corresponding to the first of the specified font families that is valid.

Implements VectSharp.IFontLibrary.

Definition at line 129 of file FontLibrary.cs.

7.52.2.4 ResolveFontFamily() [4/4]

```
virtual FontFamily VectSharp.FontLibrary.ResolveFontFamily (
            string fontFamily,
            params string[] fallback )  [virtual]
```

Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, try to instantiate the font family using the *fallback* . If none of the fallback family names or true type files are valid, an exception might be raised.

Parameters

fontFamily	The name of the font family to create, or the path to a TTF file.
fallback	Names of additional font families or TTF files, which will be tried if the first *fontFamily* is not valid.

Returns

A FontFamily object corresponding to the first of the specified font families that is valid.

Implements VectSharp.IFontLibrary.

Definition at line 76 of file FontLibrary.cs.

The documentation for this class was generated from the following file:

- VectSharp/FontLibrary.cs

7.53 VectSharp.Font.FontUnderline Class Reference

Represents options to underline text.

Properties

- bool SkipDescenders [get, set]

  Determines whether the underline skips the parts of the glyph that would intersect with it or not.
- double Position [get, set]

  Determines the position of the top of the underline with respect to the text baseline. Positive values are below the baseline, negative values are above it. This is expressed as a fraction of the font size.
- double Thickness [get, set]

  Determines the thickness of the underline, expressed as a fraction of the font size.
- LineCaps LineCap [get, set]

  Determines the caps at the start and end of the underline.
- bool FollowItalicAngle [get, set]

  Determine whether the shape of the underline is slanted to follow the angle of italic fonts.

7.53.1 Detailed Description

Represents options to underline text.

Definition at line 39 of file Font.cs.

7.53.2 Property Documentation

7.53.2.1 FollowItalicAngle

```
bool VectSharp.Font.FontUnderline.FollowItalicAngle  [get], [set]
```

Determine whether the shape of the underline is slanted to follow the angle of italic fonts.

Definition at line 64 of file Font.cs.

7.53.2.2 LineCap

```
LineCaps VectSharp.Font.FontUnderline.LineCap  [get], [set]
```

Determines the caps at the start and end of the underline.

Definition at line 59 of file Font.cs.

7.53.2.3 Position

```
double VectSharp.Font.FontUnderline.Position  [get], [set]
```

Determines the position of the top of the underline with respect to the text baseline. Positive values are below the baseline, negative values are above it. This is expressed as a fraction of the font size.

Definition at line 49 of file Font.cs.

7.53.2.4 SkipDescenders

```
bool VectSharp.Font.FontUnderline.SkipDescenders  [get], [set]
```

Determines whether the underline skips the parts of the glyph that would intersect with it or not.

Definition at line 44 of file Font.cs.

7.53.2.5 Thickness

```
double VectSharp.Font.FontUnderline.Thickness  [get], [set]
```

Determines the thickness of the underline, expressed as a fraction of the font size.

Definition at line 54 of file Font.cs.

The documentation for this class was generated from the following file:

- VectSharp/Font.cs

7.54 VectSharp.Markdown.FormattedString Struct Reference

Represents a string with associated formatting information.

Public Member Functions

- FormattedString (string text, Colour colour, bool isBold, bool isItalic)
  Creates a new FormattedString instance.

Properties

- string Text `[get]`
  The text represented by this object.
- Colour Colour `[get]`
  The colour of the text.
- bool IsBold `[get]`
  Whether the text should be rendered as bold or not.
- bool IsItalic `[get]`
  Whether the text should be rendered as italic or not.

7.54.1 Detailed Description

Represents a string with associated formatting information.

Definition at line 32 of file SyntaxHighlighting.cs.

7.54.2 Constructor & Destructor Documentation

7.54.2.1 FormattedString()

```
VectSharp.Markdown.FormattedString.FormattedString (
            string text,
            Colour colour,
            bool isBold,
            bool isItalic )
```

Creates a new FormattedString instance.

Parameters

text	The text of the object.
colour	The colour of the text.
isBold	Whether the text should be rendered as bold or not.
isItalic	Whether the text should be rendered as italic or not.

Definition at line 61 of file SyntaxHighlighting.cs.

7.54.3 Property Documentation

7.54.3.1 Colour

```
Colour VectSharp.Markdown.FormattedString.Colour  [get]
```

The colour of the text.

Definition at line 42 of file SyntaxHighlighting.cs.

7.54.3.2 IsBold

```
bool VectSharp.Markdown.FormattedString.IsBold  [get]
```

Whether the text should be rendered as bold or not.

Definition at line 47 of file SyntaxHighlighting.cs.

7.54.3.3 IsItalic

```
bool VectSharp.Markdown.FormattedString.IsItalic  [get]
```

Whether the text should be rendered as italic or not.

Definition at line 52 of file SyntaxHighlighting.cs.

7.54.3.4 Text

```
string VectSharp.Markdown.FormattedString.Text  [get]
```

The text represented by this object.

Definition at line 37 of file SyntaxHighlighting.cs.

The documentation for this struct was generated from the following file:

- VectSharp.Markdown/SyntaxHighlighting.cs

7.55 VectSharp.FormattedText Class Reference

Represents a run of text that should be drawn with the same style.

Public Member Functions

- FormattedText (string text, Font font, Script script=Script.Normal, Brush brush=null, TextSpacing? spacing=null)

  Creates a new FormattedText instance with the specified text , font , script position and brush .

Static Public Member Functions

- static IEnumerable< FormattedText > Format (string text, Font normalFont, Font boldFont, Font italicFont, Font boldItalicFont, Brush defaultBrush=null, TextSpacing? spacing=null)

  Parse the formatting information contained in a text string into a collection of FormattedText objects.
- static IEnumerable< FormattedText > Format (string text, FontFamily.StandardFontFamilies fontFamily, double fontSize, bool defaultUnderline=false, Brush defaultBrush=null, TextSpacing? spacing=null)

  Parse the formatting information contained in a text string into a collection of FormattedText objects, using fonts from a standard font family.

Properties

- string Text [get]

  Represents the text represented by this instance.
- Font Font [get]

  Represents the font that should be used to draw the text.
- Script Script [get]

  Represents the position of the text.
- Brush Brush [get]

  Represents the brush that should be used to draw the text. If this is null, the default brush is used.
- TextSpacing Spacing [get]

  Represents the text spacing parameters used to draw the text.

7.55.1 Detailed Description

Represents a run of text that should be drawn with the same style.

Definition at line 51 of file FormattedText.cs.

7.55.2 Constructor & Destructor Documentation

7.55.2.1 FormattedText()

```
VectSharp.FormattedText.FormattedText (
            string text,
            Font font,
            Script script = Script.Normal,
            Brush brush = null,
            TextSpacing?  spacing = null )
```

Creates a new FormattedText instance with the specified *text* , *font* , *script* position and *brush* .

Parameters

text	The text that will be contained in the new FormattedText.
font	The font that will be used by the new FormattedText.
script	The script position of the new FormattedText.
brush	The brush that will be used by the new FormattedText.
spacing	The text spacing parameters used to draw the text.

Definition at line 86 of file FormattedText.cs.

7.55.3 Member Function Documentation

7.55.3.1 Format() [1/2]

```
static IEnumerable< FormattedText > VectSharp.FormattedText.Format (
            string text,
            Font normalFont,
            Font boldFont,
            Font italicFont,
            Font boldItalicFont,
            Brush defaultBrush = null,
            TextSpacing?  spacing = null )  [static]
```

Parse the formatting information contained in a text string into a collection of FormattedText objects.

Parameters

text	The string containing formatting information. Format information is specified using HTML-like tags: • or are used for bold text; • or are used for text in italics; • <u></u> are used for underlined text; • and are used, respectively, for superscript and subscript text; • <#COLOUR></#> is used to specify the colour of the text, where COLOUR is a CSS colour string (e.g. <#red>, <#0080FF>, or <#rgba(128, 80, 52, 0.5)>).
normalFont	The font that will be used for text that is neither bold nor italic.
boldFont	The font that will be used for text that is bold. Note that this does not necessarily have to be a bold font; this is just the font that is applied to text contained within tags.
italicFont	The font that will be used for text that is in italics. Note that this does not necessarily have to be an italic font; this is just the font that is applied to text contained within tags.
boldItalicFont	The font that will be used for text that is both in bold and in italics.
defaultBrush	The default Brush that will be used for text runs that do not specify a colour. If this is null, the default Brush will be the one specified in the painting call.
spacing	The text spacing parameters used to draw the text.

Returns

A lazy collection of FormattedText objects. Note that every enumeration of this collection causes the text to be parsed again; if you need to enumerate this collection more than once, you should probably convert it e.g. to a List<T>.

Definition at line 114 of file FormattedText.cs.

7.55.3.2 Format() [2/2]

```
static IEnumerable< FormattedText > VectSharp.FormattedText.Format (
            string text,
            FontFamily.StandardFontFamilies fontFamily,
            double fontSize,
            bool defaultUnderline = false,
            Brush defaultBrush = null,
            TextSpacing?  spacing = null )  [static]
```

Parse the formatting information contained in a text string into a collection of FormattedText objects, using fonts from a standard font family.

Parameters

text	The string containing formatting information. Format information is specified using HTML-like tags: • `<b></b>` or `<strong></strong>` are used for bold text; • `<i></i>` or `<em></em>` are used for text in italics; • `<u></u>` are used for underlined text; • `<sup></sup>` and `<sub></sub>` are used, respectively, for superscript and subscript text; • `<#COLOUR></#>` is used to specify the colour of the text, where `COLOUR` is a CSS colour string (e.g. `<#red>`, `<#0080FF>`, or `<#rgba(128, 80, 52, 0.5)>`).
fontFamily	The font family from which the fonts will be created. If this is a regular font family, the bold, italic and bold-italic versions of the font will be used for the formatted text. Otherwise, the relevant font styles will be toggled (e.g. if the supplied font family is bold, then regular text in the formatted string will be displayed as bold, while bold text in the formatted string will be displayed as regular text).
fontSize	The size of the fonts to use.
defaultUnderline	Determines whether text should be underlined by default. This is toggled by `<u></u>` tags.
defaultBrush	The default Brush that will be used for text runs that do not specify a colour. If this is `null`, the default Brush will be the one specified in the painting call.
spacing	The text spacing parameters used to draw the text.

Returns

A lazy collection of FormattedText objects. Note that every enumeration of this collection causes the text to be parsed again; if you need to enumerate this collection more than once, you should probably convert it e.g. to a List<T>.

Definition at line 297 of file FormattedText.cs.

7.55.4 Property Documentation

7.55.4.1 Brush

```
Brush VectSharp.FormattedText.Brush  [get]
```

Represents the brush that should be used to draw the text. If this is null, the default brush is used.

Definition at line 71 of file FormattedText.cs.

7.55.4.2 Font

```
Font VectSharp.FormattedText.Font  [get]
```

Represents the font that should be used to draw the text.

Definition at line 61 of file FormattedText.cs.

7.55.4.3 Script

```
Script VectSharp.FormattedText.Script  [get]
```

Represents the position of the text.

Definition at line 66 of file FormattedText.cs.

7.55.4.4 Spacing

```
TextSpacing VectSharp.FormattedText.Spacing  [get]
```

Represents the text spacing parameters used to draw the text.

Definition at line 76 of file FormattedText.cs.

7.55.4.5 Text

```
string VectSharp.FormattedText.Text  [get]
```

Represents the text represented by this instance.

Definition at line 56 of file FormattedText.cs.

The documentation for this class was generated from the following file:

- VectSharp/FormattedText.cs

7.56 VectSharp.FormattedTextExtensions Class Reference

Contains extension methods for collections of FormattedText objects.

Static Public Member Functions

- static Font.DetailedFontMetrics Measure (this IEnumerable< FormattedText > text)

  Measures a collection of FormattedText objects.
- static string GetText (this IEnumerable< FormattedText > text)

  Extracts the text from a collection of FormattedText objects.

7.56.1 Detailed Description

Contains extension methods for collections of FormattedText objects.

Definition at line 552 of file FormattedText.cs.

7.56.2 Member Function Documentation

7.56.2.1 GetText()

```
static string VectSharp.FormattedTextExtensions.GetText (
            this IEnumerable< FormattedText > text )  [static]
```

Extracts the text from a collection of FormattedText objects.

Parameters

text	The collection of FormattedText objects whose text should be extracted.

Returns

A text rappresentation of the collection of FormattedText objects.

Definition at line 678 of file FormattedText.cs.

7.56.2.2 Measure()

```
static Font.DetailedFontMetrics VectSharp.FormattedTextExtensions.Measure (
            this IEnumerable< FormattedText > text )  [static]
```

Measures a collection of FormattedText objects.

Parameters

text	The collection of FormattedText objects to be measured.

Returns

A Font.DetailedFontMetrics containing detailed measurements for the text obtained by composing the elements in the FormattedText collection.

Definition at line 668 of file FormattedText.cs.

The documentation for this class was generated from the following file:

- VectSharp/FormattedText.cs

7.57 VectSharp.Frame Class Reference

A key frame for an animation.

Public Member Functions

- Frame (Graphics graphics, double duration)
  Creates a new Frame with the specified contents and duration.

Properties

- double Duration `[get]`
  The duration of the frame, in milliseconds.
- Graphics Graphics `[get]`
  The contents of the frame.

7.57.1 Detailed Description

A key frame for an animation.

Definition at line 1220 of file Animation.cs.

7.57.2 Constructor & Destructor Documentation

7.57.2.1 Frame()

```
VectSharp.Frame.Frame (
            Graphics graphics,
            double duration )
```

Creates a new Frame with the specified contents and duration.

Parameters

graphics	The contents of the frame.
duration	The duration of the frame, in milliseconds.

Definition at line 1237 of file Animation.cs.

7.57.3 Property Documentation

7.57.3.1 Duration

```
double VectSharp.Frame.Duration  [get]
```

The duration of the frame, in milliseconds.

Definition at line 1225 of file Animation.cs.

7.57.3.2 Graphics

```
Graphics VectSharp.Frame.Graphics  [get]
```

The contents of the frame.

Definition at line 1230 of file Animation.cs.

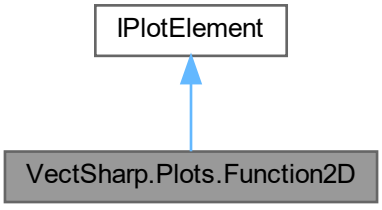
The documentation for this class was generated from the following file:

- VectSharp/Animation.cs

7.58 VectSharp.Plots.Function2D Class Reference

A plot element that plots a function of two variables.

Inheritance diagram for VectSharp.Plots.Function2D:



Public Types

- enum PlotType

  Describes the kind of plots that can be produced.

Public Member Functions

- Function2D (Function2DGrid function, IContinuousInvertibleCoordinateSystem coordinateSystem)

  Create a new Function2D instance.
- unsafe void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IDataPointElement SampledPointElement = new PathDataPointElement() `[get, set]`

  The symbol to draw at the sampled points.
- int RasterResolutionX = 512 `[get, set]`

  Resolution on the X axis for the rasterised tesselation.
- int RasterResolutionY = 512 `[get, set]`

  Resolution on the Y axis for the rasterised tesselation.
- Function2DGrid Function `[get, set]`

  The function to plot.
- PlotType Type = PlotType.SampledPoints `[get, set]`

  The kind of plot that is produced.
- Func< double, Colour > Colouring = x => { x = Math.Max(0, Math.Min(1, x)); return Colour.FromRgb(x, x, x); } `[get, set]`

  A function associating sampled function values to a Colour. You should set this to a function accepting a single `double` argument ranging between 0 and 1, and returning the corresponding colour. The default value returns black 0 and white for 1.
- string Tag `[get, set]`

  A tag to identify the function in the plot.
- IContinuousInvertibleCoordinateSystem CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.

7.58.1 Detailed Description

A plot element that plots a function of two variables.

Definition at line 520 of file Function2D.cs.

7.58.2 Member Enumeration Documentation

7.58.2.1 PlotType

```
enum VectSharp.Plots.Function2D.PlotType
```

Describes the kind of plots that can be produced.

Definition at line 525 of file Function2D.cs.

7.58.3 Constructor & Destructor Documentation

7.58.3.1 Function2D()

```
VectSharp.Plots.Function2D.Function2D (
            Function2DGrid function,
            IContinuousInvertibleCoordinateSystem coordinateSystem )
```

Create a new Function2D instance.

Parameters

function	The function to plot.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 594 of file Function2D.cs.

7.58.4 Member Function Documentation

7.58.4.1 Plot()

```
unsafe void VectSharp.Plots.Function2D.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 601 of file Function2D.cs.

7.58.5 Property Documentation

7.58.5.1 Colouring

```
Func<double, Colour> VectSharp.Plots.Function2D.Colouring = x => { x = Math.Max(0, Math.↩
Min(1, x)); return Colour.FromRgb(x, x, x); }  [get], [set]
```

A function associating sampled function values to a Colour. You should set this to a function accepting a single `double` argument ranging between 0 and 1, and returning the corresponding colour. The default value returns black 0 and white for 1.

Definition at line 576 of file Function2D.cs.

7.58.5.2 CoordinateSystem

```
IContinuousInvertibleCoordinateSystem VectSharp.Plots.Function2D.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 586 of file Function2D.cs.

7.58.5.3 Function

```
Function2DGrid VectSharp.Plots.Function2D.Function  [get], [set]
```

The function to plot.

Definition at line 564 of file Function2D.cs.

7.58.5.4 RasterResolutionX

```
int VectSharp.Plots.Function2D.RasterResolutionX = 512  [get], [set]
```

Resolution on the X axis for the rasterised tesselation.

Definition at line 554 of file Function2D.cs.

7.58.5.5 RasterResolutionY

```
int VectSharp.Plots.Function2D.RasterResolutionY = 512  [get], [set]
```

Resolution on the Y axis for the rasterised tesselation.

Definition at line 559 of file Function2D.cs.

7.58.5.6 SampledPointElement

```
IDataPointElement VectSharp.Plots.Function2D.SampledPointElement = new PathDataPointElement()
[get], [set]
```

The symbol to draw at the sampled points.

Definition at line 549 of file Function2D.cs.

7.58.5.7 Tag

```
string VectSharp.Plots.Function2D.Tag  [get], [set]
```

A tag to identify the function in the plot.

Definition at line 581 of file Function2D.cs.

7.58.5.8 Type

```
PlotType VectSharp.Plots.Function2D.Type = PlotType.SampledPoints  [get], [set]
```

The kind of plot that is produced.

Definition at line 569 of file Function2D.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/Function2D.cs

7.59 VectSharp.Plots.Function2DGrid Class Reference

Represents a function of two variables that has been sampled in some points.

Public Types

- enum GridType

  Describes the arrangement of the points that have been sampled.

Public Member Functions

- Function2DGrid (IReadOnlyList< IReadOnlyList< double > > dataPoints)

  Create a new Function2DGrid from a list of sampled values.
- Function2DGrid (Func< double[ ], double > function, double minX, double minY, double maxX, double maxY, int stepsX, int stepsY, GridType type)

  Create a new Function2DGrid by sampling the provided function.
- Function2DGrid ToRectangular ()

  Converts a hexagonal grid into a rectangular grid.

Properties

- IReadOnlyList< IReadOnlyList< double > > DataPoints `[get]`

  The points where the function has been sampled.
- double MinX `[get]`

  The minimum X value that has been sampled.
- double MaxX `[get]`

  The maximum X value that has been sampled.
- double MinY `[get]`

  The minimum Y value that has been sampled.
- double MaxY `[get]`

  The maximum Y value that has been sampled.
- double MinZ `[get]`

  The minimum value that has been obtained for the function.
- double MaxZ `[get]`

  The maximum value that has been obtained for the function.
- int StepsX `[get]`

  If the function has been sampled along a regular grid, the number of steps between MinX and MaxX on the X axis.
- int StepsY `[get]`

  If the function has been sampled along a regular grid, the number of steps between MinY and MaxY on the X axis.
- GridType Type `[get]`

  The type of grid.

7.59.1 Detailed Description

Represents a function of two variables that has been sampled in some points.

Definition at line 29 of file Function2D.cs.

7.59.2 Member Enumeration Documentation

7.59.2.1 GridType

```
enum VectSharp.Plots.Function2DGrid.GridType
```

Describes the arrangement of the points that have been sampled.

Definition at line 34 of file Function2D.cs.

7.59.3 Constructor & Destructor Documentation

7.59.3.1 Function2DGrid() [1/2]

```
VectSharp.Plots.Function2DGrid.Function2DGrid (
            IReadOnlyList< IReadOnlyList< double > > dataPoints )
```

Create a new Function2DGrid from a list of sampled values.

Parameters

dataPoints	The sampled values. Each element of this IReadOnlyList<T> should be a collection of three values: the X coordinate of the sampled point, the Y coordinate of the sampled point, and the value of the function at that point.

Definition at line 117 of file Function2D.cs.

7.59.3.2 Function2DGrid() [2/2]

```
VectSharp.Plots.Function2DGrid.Function2DGrid (
            Func< double[], double > function,
            double minX,
            double minY,
            double maxX,
            double maxY,
            int stepsX,
            int stepsY,
            GridType type )
```

Create a new Function2DGrid by sampling the provided function.

Parameters

function	The function to sample.
minX	The minimum X value at which the function should be sampled.
minY	The minimum Y value at which the function should be sampled.
maxX	The maximum X value at which the function should be sampled.
maxY	The maximum Y value at which the function should be sampled.

Parameters

stepsX	If *type* is not GridType.Irregular, the number of steps between *minX* and *maxX* on the X axis. Otherwise, the number of sampled points is determined by multiplying *stepsX* and *stepsY* together.
stepsY	If *type* is not GridType.Irregular, the number of steps between *minY* and *maxY* on the Y axis. Otherwise, the number of sampled points is determined by multiplying *stepsX* and *stepsY* together.
type	The strategy used to select points to sample. If this is GridType.Irregular, uniformly distributed random points between (*minX* , *minY*) and (*maxX* , *maxY*) are sampled.

Definition at line 163 of file Function2D.cs.

7.59.4 Member Function Documentation

7.59.4.1 ToRectangular()

```
Function2DGrid VectSharp.Plots.Function2DGrid.ToRectangular ( )
```

Converts a hexagonal grid into a rectangular grid.

Returns

If Type is GridType.Rectangular, this method returns the current instance.
If Type is GridType.HexagonHorizontal or GridType.HexagonVertical, a new Function2DGrid with Type equal to GridType.Rectangular is returned. The sampled points in this grid are obtained by performing a bilinear interpolation on the sampled points from this grid. The returned grid will always be "denser" than the current instance.
If Type is GridType.Irregular, an InvalidOperationException is thrown.

Exceptions

InvalidOperationException	Thrown if Type is GridType.Irregular.

Definition at line 322 of file Function2D.cs.

7.59.5 Property Documentation

7.59.5.1 DataPoints

```
IReadOnlyList<IReadOnlyList<double> > VectSharp.Plots.Function2DGrid.DataPoints [get]
```

The points where the function has been sampled.

Definition at line 62 of file Function2D.cs.

7.59.5.2 MaxX

```
double VectSharp.Plots.Function2DGrid.MaxX  [get]
```

The maximum X value that has been sampled.

Definition at line 72 of file Function2D.cs.

7.59.5.3 MaxY

```
double VectSharp.Plots.Function2DGrid.MaxY  [get]
```

The maximum Y value that has been sampled.

Definition at line 82 of file Function2D.cs.

7.59.5.4 MaxZ

```
double VectSharp.Plots.Function2DGrid.MaxZ  [get]
```

The maximum value that has been obtained for the function.

Definition at line 92 of file Function2D.cs.

7.59.5.5 MinX

```
double VectSharp.Plots.Function2DGrid.MinX  [get]
```

The minimum X value that has been sampled.

Definition at line 67 of file Function2D.cs.

7.59.5.6 MinY

```
double VectSharp.Plots.Function2DGrid.MinY  [get]
```

The minimum Y value that has been sampled.

Definition at line 77 of file Function2D.cs.

7.59.5.7 MinZ

```
double VectSharp.Plots.Function2DGrid.MinZ  [get]
```

The minimum value that has been obtained for the function.

Definition at line 87 of file Function2D.cs.

7.59.5.8 StepsX

```
int VectSharp.Plots.Function2DGrid.StepsX  [get]
```

If the function has been sampled along a regular grid, the number of steps between MinX and MaxX on the X axis.

Definition at line 98 of file Function2D.cs.

7.59.5.9 StepsY

```
int VectSharp.Plots.Function2DGrid.StepsY  [get]
```

If the function has been sampled along a regular grid, the number of steps between MinY and MaxY on the X axis.

Definition at line 104 of file Function2D.cs.

7.59.5.10 Type

```
GridType VectSharp.Plots.Function2DGrid.Type  [get]
```

The type of grid.

Definition at line 109 of file Function2D.cs.

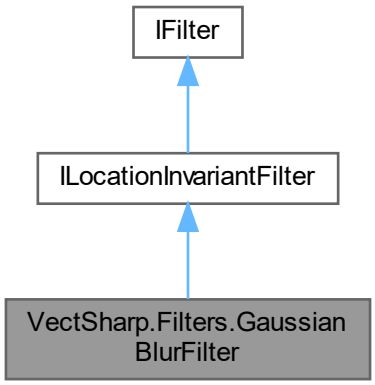
The documentation for this class was generated from the following file:

- VectSharp.Plots/Function2D.cs

7.60 VectSharp.Filters.GaussianBlurFilter Class Reference

Represents a filter that applies a Gaussian blur effect.

Inheritance diagram for VectSharp.Filters.GaussianBlurFilter:



Public Member Functions

- GaussianBlurFilter (double standardDeviation)

  Creates a new GaussianBlurFilter with the specified standard deviation.
- RasterImage Filter (RasterImage image, double scale)

  Applies the filter to a RasterImage.

Parameters

image	*The RasterImage to which the filter will be applied.*
scale	*The scale of the image with respect to the filter.*

Returns

A new RasterImage containing the filtered image. The source image is left unaltered.

Properties

- double StandardDeviation `[get]`

  The standard deviation of the Gaussian blur.
- Point TopLeftMargin `[get]`

  Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.
- Point BottomRightMargin `[get]`

  Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.

7.60.1 Detailed Description

Represents a filter that applies a Gaussian blur effect.

Definition at line 27 of file GaussianBlurFilter.cs.

7.60.2 Constructor & Destructor Documentation

7.60.2.1 GaussianBlurFilter()

```
VectSharp.Filters.GaussianBlurFilter.GaussianBlurFilter (
            double standardDeviation )
```

Creates a new GaussianBlurFilter with the specified standard deviation.

Parameters

standardDeviation	The standard deviation of the Gaussian blur.

Definition at line 44 of file GaussianBlurFilter.cs.

7.60.3 Member Function Documentation

7.60.3.1 Filter()

```
RasterImage VectSharp.Filters.GaussianBlurFilter.Filter (
            RasterImage image,
            double scale )
```

Applies the filter to a RasterImage.

Parameters

image	The RasterImage to which the filter will be applied.
scale	The scale of the image with respect to the filter.

Returns

A new RasterImage containing the filtered image. The source *image* is left unaltered.

Implements VectSharp.Filters.ILocationInvariantFilter.

Definition at line 53 of file GaussianBlurFilter.cs.

7.60.4 Property Documentation

7.60.4.1 BottomRightMargin

```
Point VectSharp.Filters.GaussianBlurFilter.BottomRightMargin  [get]
```

Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.

Implements VectSharp.Filters.IFilter.

Definition at line 38 of file GaussianBlurFilter.cs.

7.60.4.2 StandardDeviation

```
double VectSharp.Filters.GaussianBlurFilter.StandardDeviation  [get]
```

The standard deviation of the Gaussian blur.

Definition at line 32 of file GaussianBlurFilter.cs.

7.60.4.3 TopLeftMargin

```
Point VectSharp.Filters.GaussianBlurFilter.TopLeftMargin  [get]
```

Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.

Implements VectSharp.Filters.IFilter.

Definition at line 35 of file GaussianBlurFilter.cs.

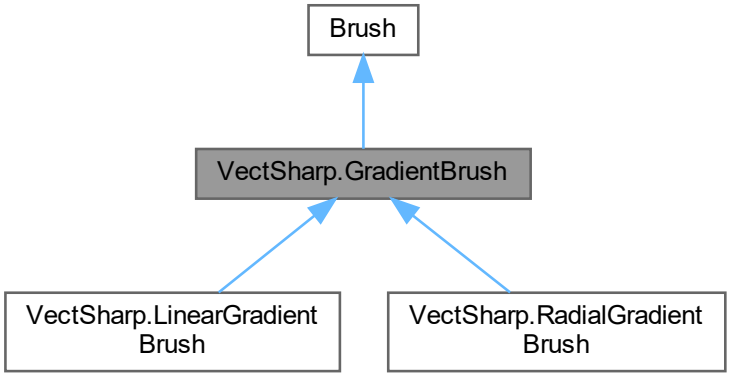
The documentation for this class was generated from the following file:

- VectSharp/Filters/GaussianBlurFilter.cs

7.61 VectSharp.GradientBrush Class Reference

Represents a brush painting with a gradient.

Inheritance diagram for VectSharp.GradientBrush:



Properties

- GradientStops GradientStops [get]

  The colour stops in the gradient.

Additional Inherited Members

7.61.1 Detailed Description

Represents a brush painting with a gradient.

Definition at line 277 of file Brush.cs.

7.61.2 Property Documentation

7.61.2.1 GradientStops

```
GradientStops VectSharp.GradientBrush.GradientStops [get]
```

The colour stops in the gradient.

Definition at line 282 of file Brush.cs.

The documentation for this class was generated from the following file:

- VectSharp/Brush.cs

7.62 VectSharp.Gradients Class Reference

Standard gradients.

Static Public Member Functions

- static Colour MagmaColouring (double x)

  Magma colour map, based on the homonymous colour scale included in the "viridis" R package ().
- static Colour InfernoColouring (double x)

  Inferno colour map, based on the homonymous colour scale included in the "viridis" R package ().
- static Colour PlasmaColouring (double x)

  Plasma colour map, based on the homonymous colour scale included in the "viridis" R package ().
- static Colour ViridisColouring (double x)

  Viridis colour map, based on the homonymous colour scale included in the "viridis" R package ().
- static Colour CividisColouring (double x)

  Cividis colour map, based on the homonymous colour scale included in the "viridis" R package ().
- static Colour RocketColouring (double x)

  Rocket colour map, based on the homonymous colour scale included in the "viridis" R package ().
- static Colour MakoColouring (double x)

  Mako colour map, based on the homonymous colour scale included in the "viridis" R package ().
- static Colour TurboColouring (double x)

  Turbo colour map, based on the homonymous colour scale included in the "viridis" R package ().

Static Public Attributes

- static readonly GradientStops TransparentToBlack = new GradientStops(new GradientStop(Colour.FromRgba(0, 0, 0, 0), 0), new GradientStop(Colour.FromRgb(0, 0, 0), 1))

  Gradient from transparent black (0) to opaque black (1).
- static readonly GradientStops WhiteToBlack = new GradientStops(new GradientStop(Colour.FromRgb(255, 255, 255), 0), new GradientStop(Colour.FromRgb(0, 0, 0), 1))

  Gradient from white (0) to black (1).
- static readonly GradientStops RedToGreen = new GradientStops(new GradientStop(Colour.FromRgb(237, 28, 36), 0), new GradientStop(Colour.FromRgb(34, 177, 76), 1))

  Gradient from red (0) to green (1).
- static readonly GradientStops Rainbow = new GradientStops(new GradientStop(Colour.FromRgb(237, 28, 36), 0), new GradientStop(Colour.FromRgb(255, 127, 39), 1.0 / 6), new GradientStop(Colour.FromRgb(255, 242, 0), 1.0 / 3), new GradientStop(Colour.FromRgb(34, 177, 76), 0.5), new GradientStop(Colour.FromRgb(0, 162, 232), 2.0 / 3), new GradientStop(Colour.FromRgb(63, 72, 204), 5.0 / 6), new GradientStop(Colour.FromRgb(163, 73, 164), 1))

  Rainbow gradient (red, orange, yellow, green, blue, indigo, violet).
- static readonly GradientStops RedYellowGreen = new GradientStops(new GradientStop(Colour.FromRgb(237, 28, 36), 0), new GradientStop(Colour.FromRgb(255, 242, 0), 0.5), new GradientStop(Colour.FromRgb(34, 177, 76), 1))

  Gradient from red (0) to yellow (0.5) to green (1).
- static readonly GradientStops OkabeItoRainbow = new GradientStops(new GradientStop(Colour.FromRgb(204, 121, 167), 0), new GradientStop(Colour.FromRgb(213, 94, 0), 1.0 / 6), new GradientStop(Colour.FromRgb(230, 159, 0), 1.0 / 3), new GradientStop(Colour.FromRgb(240, 228, 66), 0.5), new GradientStop(Colour.FromRgb(0, 158, 115), 2.0 / 3), new GradientStop(Colour.FromRgb(0, 114, 178), 5.0 / 6), new GradientStop(Colour.FromRgb(86, 180, 233), 1))

  Rainbow gradient with Okabe-Ito colour-blind safe colours ().

- static readonly GradientStops OkabeItoRainbowDiscrete = new GradientStops(new GradientStop(Colour.FromRgb(204, 121, 167), 0), new GradientStop(Colour.FromRgb(204, 121, 167), 0.2), new GradientStop(Colour.FromRgb(230, 159, 0), 0.201), new GradientStop(Colour.FromRgb(230, 159, 0), 0.4), new GradientStop(Colour.FromRgb(0, 158, 115), 0.401), new GradientStop(Colour.FromRgb(0, 158, 115), 0.6), new GradientStop(Colour.FromRgb(0, 114, 178), 0.601), new GradientStop(Colour.FromRgb(0, 114, 178), 0.8), new GradientStop(Colour.FromRgb(86, 180, 233), 0.801), new GradientStop(Colour.FromRgb(86, 180, 233), 1))

  Rainbow gradient with Okabe-Ito colour-blind safe colours () in discrete steps.
- static readonly GradientStops MutedRainbow = new GradientStops(new GradientStop(Colour.FromRgb(221, 204, 119), 0), new GradientStop(Colour.FromRgb(153, 153, 51), 0.125), new GradientStop(Colour.FromRgb(17, 119, 51), 0.25), new GradientStop(Colour.FromRgb(68, 170, 153), 0.375), new GradientStop(Colour.FromRgb(136, 204, 238), 0.5), new GradientStop(Colour.FromRgb(51, 34, 136), 0.625), new GradientStop(Colour.FromRgb(170, 68, 153), 0.75), new GradientStop(Colour.FromRgb(136, 34, 85), 0.875), new GradientStop(Colour.FromRgb(204, 102, 119), 1))

  Rainbow gradient with Paul Tol's Muted palette ().
- static readonly GradientStops MutedRainbowDiscrete = new GradientStops(new GradientStop(Colour.FromRgb(187, 187, 187), 0.101), new GradientStop(Colour.FromRgb(187, 187, 187), 0.1), new GradientStop(Colour.FromRgb(221, 204, 119), 0.101), new GradientStop(Colour.FromRgb(221, 204, 119), 0.2), new GradientStop(Colour.FromRgb(153, 153, 51), 0.201), new GradientStop(Colour.FromRgb(153, 153, 51), 0.3), new GradientStop(Colour.FromRgb(17, 119, 51), 0.301), new GradientStop(Colour.FromRgb(17, 119, 51), 0.4), new GradientStop(Colour.FromRgb(68, 170, 153), 0.401), new GradientStop(Colour.FromRgb(68, 170, 153), 0.5), new GradientStop(Colour.FromRgb(136, 204, 238), 0.501), new GradientStop(Colour.FromRgb(136, 204, 238), 0.6), new GradientStop(Colour.FromRgb(51, 34, 136), 0.601), new GradientStop(Colour.FromRgb(51, 34, 136), 0.7), new GradientStop(Colour.FromRgb(170, 68, 153), 0.701), new GradientStop(Colour.FromRgb(170, 68, 153), 0.8), new GradientStop(Colour.FromRgb(136, 34, 85), 0.801), new GradientStop(Colour.FromRgb(136, 34, 85), 0.9), new GradientStop(Colour.FromRgb(204, 102, 119), 0.901), new GradientStop(Colour.FromRgb(204, 102, 119), 1))

  Rainbow gradient with Paul Tol's Muted palette () in discrete steps.
- static readonly GradientStops Magma = new GradientStops(from el in Enumerable.Range(0, 21) let ind = (int)Math.Round(12.75 ∗ el) select new GradientStop(Colour.FromRgb(MagmaRGB[ind, 0], MagmaRGB[ind, 1], MagmaRGB[ind, 2]), el ∗ 0.05))

  Magma gradient, based on the homonymous colour scale included in the "viridis" R package ().
- static readonly GradientStops Inferno = new GradientStops(from el in Enumerable.Range(0, 21) let ind = (int)Math.Round(12.75 ∗ el) select new GradientStop(Colour.FromRgb(InfernoRGB[ind, 0], InfernoRGB[ind, 1], InfernoRGB[ind, 2]), el ∗ 0.05))

  Inferno gradient, based on the homonymous colour scale included in the "viridis" R package ().
- static readonly GradientStops Plasma = new GradientStops(from el in Enumerable.Range(0, 21) let ind = (int)Math.Round(12.75 ∗ el) select new GradientStop(Colour.FromRgb(PlasmaRGB[ind, 0], PlasmaRGB[ind, 1], PlasmaRGB[ind, 2]), el ∗ 0.05))

  Plasma gradient, based on the homonymous colour scale included in the "viridis" R package ().
- static readonly GradientStops Viridis = new GradientStops(from el in Enumerable.Range(0, 21) let ind = (int)Math.Round(12.75 ∗ el) select new GradientStop(Colour.FromRgb(ViridisRGB[ind, 0], ViridisRGB[ind, 1], ViridisRGB[ind, 2]), el ∗ 0.05))

  Viridis gradient, based on the homonymous colour scale included in the "viridis" R package ().
- static readonly GradientStops Cividis = new GradientStops(from el in Enumerable.Range(0, 21) let ind = (int)Math.Round(12.75 ∗ el) select new GradientStop(Colour.FromRgb(CividisRGB[ind, 0], CividisRGB[ind, 1], CividisRGB[ind, 2]), el ∗ 0.05))

  Cividis gradient, based on the homonymous colour scale included in the "viridis" R package ().
- static readonly GradientStops Rocket = new GradientStops(from el in Enumerable.Range(0, 21) let ind = (int)Math.Round(12.75 ∗ el) select new GradientStop(Colour.FromRgb(RocketRGB[ind, 0], RocketRGB[ind, 1], RocketRGB[ind, 2]), el ∗ 0.05))

  Rocket gradient, based on the homonymous colour scale included in the "viridis" R package ().
- static readonly GradientStops Mako = new GradientStops(from el in Enumerable.Range(0, 21) let ind = (int)Math.Round(12.75 ∗ el) select new GradientStop(Colour.FromRgb(MakoRGB[ind, 0], MakoRGB[ind, 1], MakoRGB[ind, 2]), el ∗ 0.05))

  Mako gradient, based on the homonymous colour scale included in the "viridis" R package ().

- static readonly GradientStops Turbo = new GradientStops(from el in Enumerable.Range(0, 21) let ind = (int)Math.Round(12.75 ∗ el) select new GradientStop(Colour.FromRgb(TurboRGB[ind, 0], TurboRGB[ind, 1], TurboRGB[ind, 2]), el ∗ 0.05))

  Turbo gradient, based on the homonymous colour scale included in the "viridis" R package ().

7.62.1 Detailed Description

Standard gradients.

Definition at line 13 of file Gradients.cs.

7.62.2 Member Function Documentation

7.62.2.1 CividisColouring()

```
static Colour VectSharp.Gradients.CividisColouring (
            double x )  [static]
```

Cividis colour map, based on the homonymous colour scale included in the "viridis" R package ().

Parameters

x	The position in the map (ranging from 0 to 1).

Returns

The colour at the corresponding position in the colour map.

Definition at line 162 of file Gradients.cs.

7.62.2.2 InfernoColouring()

```
static Colour VectSharp.Gradients.InfernoColouring (
            double x )  [static]
```

Inferno colour map, based on the homonymous colour scale included in the "viridis" R package ().

Parameters

x	The position in the map (ranging from 0 to 1).

Returns

The colour at the corresponding position in the colour map.

Definition at line 126 of file Gradients.cs.

7.62.2.3 MagmaColouring()

```
static Colour VectSharp.Gradients.MagmaColouring (
            double x )  [static]
```

Magma colour map, based on the homonymous colour scale included in the "viridis" R package ().

Parameters

x	The position in the map (ranging from 0 to 1).

Returns

The colour at the corresponding position in the colour map.

Definition at line 114 of file Gradients.cs.

7.62.2.4 MakoColouring()

```
static Colour VectSharp.Gradients.MakoColouring (
            double x )  [static]
```

Mako colour map, based on the homonymous colour scale included in the "viridis" R package ().

Parameters

x	The position in the map (ranging from 0 to 1).

Returns

The colour at the corresponding position in the colour map.

Definition at line 186 of file Gradients.cs.

7.62.2.5 PlasmaColouring()

```
static Colour VectSharp.Gradients.PlasmaColouring (
            double x )  [static]
```

Plasma colour map, based on the homonymous colour scale included in the "viridis" R package ().

Parameters

x	The position in the map (ranging from 0 to 1).

Returns

The colour at the corresponding position in the colour map.

Definition at line 138 of file Gradients.cs.

7.62.2.6 RocketColouring()

```
static Colour VectSharp.Gradients.RocketColouring (
            double x )  [static]
```

Rocket colour map, based on the homonymous colour scale included in the "viridis" R package ().

Parameters

x	The position in the map (ranging from 0 to 1).

Returns

The colour at the corresponding position in the colour map.

Definition at line 174 of file Gradients.cs.

7.62.2.7 TurboColouring()

```
static Colour VectSharp.Gradients.TurboColouring (
            double x )  [static]
```

Turbo colour map, based on the homonymous colour scale included in the "viridis" R package ().

Parameters

x	The position in the map (ranging from 0 to 1).

Returns

The colour at the corresponding position in the colour map.

Definition at line 198 of file Gradients.cs.

7.62.2.8 ViridisColouring()

```
static Colour VectSharp.Gradients.ViridisColouring (
            double x )  [static]
```

Viridis colour map, based on the homonymous colour scale included in the "viridis" R package ().

Parameters

x	The position in the map (ranging from 0 to 1).

Returns

The colour at the corresponding position in the colour map.

Definition at line 150 of file Gradients.cs.

7.62.3 Member Data Documentation

7.62.3.1 Cividis

```
readonly GradientStops VectSharp.Gradients.Cividis = new GradientStops(from el in Enumerable.↩
Range(0, 21) let ind = (int)Math.Round(12.75 * el) select new GradientStop(Colour.FromRgb(Cividis↩
RGB[ind, 0], CividisRGB[ind, 1], CividisRGB[ind, 2]), el * 0.05))  [static]
```

Cividis gradient, based on the homonymous colour scale included in the "viridis" R package ().

Definition at line 92 of file Gradients.cs.

7.62.3.2 Inferno

```
readonly GradientStops VectSharp.Gradients.Inferno = new GradientStops(from el in Enumerable.↩
Range(0, 21) let ind = (int)Math.Round(12.75 * el) select new GradientStop(Colour.FromRgb(Inferno↩
RGB[ind, 0], InfernoRGB[ind, 1], InfernoRGB[ind, 2]), el * 0.05))  [static]
```

Inferno gradient, based on the homonymous colour scale included in the "viridis" R package ().

Definition at line 77 of file Gradients.cs.

7.62.3.3 Magma

```
readonly GradientStops VectSharp.Gradients.Magma = new GradientStops(from el in Enumerable.↩
Range(0, 21) let ind = (int)Math.Round(12.75 * el) select new GradientStop(Colour.FromRgb(Magma↩
RGB[ind, 0], MagmaRGB[ind, 1], MagmaRGB[ind, 2]), el * 0.05))  [static]
```

Magma gradient, based on the homonymous colour scale included in the "viridis" R package ().

Definition at line 72 of file Gradients.cs.

7.62.3.4 Mako

```
readonly GradientStops VectSharp.Gradients.Mako = new GradientStops(from el in Enumerable.↩
Range(0, 21) let ind = (int)Math.Round(12.75 * el) select new GradientStop(Colour.FromRgb(Mako↩
RGB[ind, 0], MakoRGB[ind, 1], MakoRGB[ind, 2]), el * 0.05))  [static]
```

Mako gradient, based on the homonymous colour scale included in the "viridis" R package ().

Definition at line 102 of file Gradients.cs.

7.62.3.5 MutedRainbow

```
readonly GradientStops VectSharp.Gradients.MutedRainbow = new GradientStops(new GradientStop(Colour.FromRgb(22
204, 119), 0), new GradientStop(Colour.FromRgb(153, 153, 51), 0.125), new GradientStop(Colour.FromRgb(17,
119, 51), 0.25), new GradientStop(Colour.FromRgb(68, 170, 153), 0.375), new GradientStop(Colour.FromRgb(136,
204, 238), 0.5), new GradientStop(Colour.FromRgb(51, 34, 136), 0.625), new GradientStop(Colour.FromRgb(170,
68, 153), 0.75), new GradientStop(Colour.FromRgb(136, 34, 85), 0.875), new GradientStop(Colour.FromRgb(204,
102, 119), 1))  [static]
```

Rainbow gradient with Paul Tol's Muted palette ().

Definition at line 53 of file Gradients.cs.

7.62.3.6 MutedRainbowDiscrete

```
readonly GradientStops VectSharp.Gradients.MutedRainbowDiscrete = new GradientStops(new GradientStop(Colour.Fr
187, 187), 0.101), new GradientStop(Colour.FromRgb(187, 187, 187), 0.1), new GradientStop(Colour.FromRgb(221,
204, 119), 0.101), new GradientStop(Colour.FromRgb(221, 204, 119), 0.2), new GradientStop(Colour.FromRgb(153,
153, 51), 0.201), new GradientStop(Colour.FromRgb(153, 153, 51), 0.3), new GradientStop(Colour.FromRgb(17,
119, 51), 0.301), new GradientStop(Colour.FromRgb(17, 119, 51), 0.4), new GradientStop(Colour.FromRgb(68,
170, 153), 0.401), new GradientStop(Colour.FromRgb(68, 170, 153), 0.5), new GradientStop(Colour.FromRgb(136,
204, 238), 0.501), new GradientStop(Colour.FromRgb(136, 204, 238), 0.6), new GradientStop(Colour.FromRgb(51,
34, 136), 0.601), new GradientStop(Colour.FromRgb(51, 34, 136), 0.7), new GradientStop(Colour.FromRgb(170,
68, 153), 0.701), new GradientStop(Colour.FromRgb(170, 68, 153), 0.8), new GradientStop(Colour.FromRgb(136,
34, 85), 0.801), new GradientStop(Colour.FromRgb(136, 34, 85), 0.9), new GradientStop(Colour.FromRgb(204,
102, 119), 0.901), new GradientStop(Colour.FromRgb(204, 102, 119), 1))  [static]
```

Rainbow gradient with Paul Tol's Muted palette () in discrete steps.

Definition at line 58 of file Gradients.cs.

7.62.3.7 OkabeItoRainbow

```
readonly GradientStops VectSharp.Gradients.OkabeItoRainbow = new GradientStops(new GradientStop(Colour.FromRgb
121, 167), 0), new GradientStop(Colour.FromRgb(213, 94, 0), 1.0 / 6), new GradientStop(Colour.FromRgb(230,
159, 0), 1.0 / 3), new GradientStop(Colour.FromRgb(240, 228, 66), 0.5), new GradientStop(Colour.FromRgb(0,
158, 115), 2.0 / 3), new GradientStop(Colour.FromRgb(0, 114, 178), 5.0 / 6), new GradientStop(Colour.FromRgb(8
180, 233), 1))  [static]
```

Rainbow gradient with Okabe-Ito colour-blind safe colours ().

Definition at line 43 of file Gradients.cs.

7.62.3.8 OkabeItoRainbowDiscrete

```
readonly GradientStops VectSharp.Gradients.OkabeItoRainbowDiscrete = new GradientStops(new
GradientStop(Colour.FromRgb(204, 121, 167), 0), new GradientStop(Colour.FromRgb(204, 121,
167), 0.2), new GradientStop(Colour.FromRgb(230, 159, 0), 0.201), new GradientStop(Colour.FromRgb(230,
159, 0), 0.4), new GradientStop(Colour.FromRgb(0, 158, 115), 0.401), new GradientStop(Colour.FromRgb(0,
158, 115), 0.6), new GradientStop(Colour.FromRgb(0, 114, 178), 0.601), new GradientStop(Colour.FromRgb(0,
114, 178), 0.8), new GradientStop(Colour.FromRgb(86, 180, 233), 0.801), new GradientStop(Colour.FromRgb(86,
180, 233), 1))  [static]
```

Rainbow gradient with Okabe-Ito colour-blind safe colours () in discrete steps.

Definition at line 48 of file Gradients.cs.

7.62.3.9 Plasma

```
readonly GradientStops VectSharp.Gradients.Plasma = new GradientStops(from el in Enumerable.↩
Range(0, 21) let ind = (int)Math.Round(12.75 * el) select new GradientStop(Colour.FromRgb(Plasma↩
RGB[ind, 0], PlasmaRGB[ind, 1], PlasmaRGB[ind, 2]), el * 0.05))  [static]
```

Plasma gradient, based on the homonymous colour scale included in the "viridis" R package ().

Definition at line 82 of file Gradients.cs.

7.62.3.10 Rainbow

```
readonly GradientStops VectSharp.Gradients.Rainbow = new GradientStops(new GradientStop(Colour.FromRgb(237,
28, 36), 0), new GradientStop(Colour.FromRgb(255, 127, 39), 1.0 / 6), new GradientStop(Colour.FromRgb(255,
242, 0), 1.0 / 3), new GradientStop(Colour.FromRgb(34, 177, 76), 0.5), new GradientStop(Colour.FromRgb(0,
162, 232), 2.0 / 3), new GradientStop(Colour.FromRgb(63, 72, 204), 5.0 / 6), new GradientStop(Colour.FromRgb(1
73, 164), 1))  [static]
```

Rainbow gradient (red, orange, yellow, green, blue, indigo, violet).

Definition at line 33 of file Gradients.cs.

7.62.3.11 RedToGreen

```
readonly GradientStops VectSharp.Gradients.RedToGreen = new GradientStops(new GradientStop(Colour.FromRgb(237,
28, 36), 0), new GradientStop(Colour.FromRgb(34, 177, 76), 1))  [static]
```

Gradient from red (0) to green (1).

Definition at line 28 of file Gradients.cs.

7.62.3.12 RedYellowGreen

```
readonly GradientStops VectSharp.Gradients.RedYellowGreen = new GradientStops(new GradientStop(Colour.FromRgb(
28, 36), 0), new GradientStop(Colour.FromRgb(255, 242, 0), 0.5), new GradientStop(Colour.FromRgb(34,
177, 76), 1))  [static]
```

Gradient from red (0) to yellow (0.5) to green (1).

Definition at line 38 of file Gradients.cs.

7.62.3.13 Rocket

```
readonly GradientStops VectSharp.Gradients.Rocket = new GradientStops(from el in Enumerable.↩
Range(0, 21) let ind = (int)Math.Round(12.75 * el) select new GradientStop(Colour.FromRgb(Rocket↩
RGB[ind, 0], RocketRGB[ind, 1], RocketRGB[ind, 2]), el * 0.05))  [static]
```

Rocket gradient, based on the homonymous colour scale included in the "viridis" R package ().

Definition at line 97 of file Gradients.cs.

7.62.3.14 TransparentToBlack

```
readonly GradientStops VectSharp.Gradients.TransparentToBlack = new GradientStops(new GradientStop(Colour.From
0, 0, 0), 0), new GradientStop(Colour.FromRgb(0, 0, 0), 1))  [static]
```

Gradient from transparent black (0) to opaque black (1).

Definition at line 18 of file Gradients.cs.

7.62.3.15 Turbo

```
readonly GradientStops VectSharp.Gradients.Turbo = new GradientStops(from el in Enumerable.↩
Range(0, 21) let ind = (int)Math.Round(12.75 * el) select new GradientStop(Colour.FromRgb(Turbo↩
RGB[ind, 0], TurboRGB[ind, 1], TurboRGB[ind, 2]), el * 0.05))  [static]
```

Turbo gradient, based on the homonymous colour scale included in the "viridis" R package ().

Definition at line 107 of file Gradients.cs.

7.62.3.16 Viridis

```
readonly GradientStops VectSharp.Gradients.Viridis = new GradientStops(from el in Enumerable.↩
Range(0, 21) let ind = (int)Math.Round(12.75 * el) select new GradientStop(Colour.FromRgb(Viridis↩
RGB[ind, 0], ViridisRGB[ind, 1], ViridisRGB[ind, 2]), el * 0.05))  [static]
```

Viridis gradient, based on the homonymous colour scale included in the "viridis" R package ().

Definition at line 87 of file Gradients.cs.

7.62.3.17 WhiteToBlack

```
readonly GradientStops VectSharp.Gradients.WhiteToBlack = new GradientStops(new GradientStop(Colour.FromRgb(25
255, 255), 0), new GradientStop(Colour.FromRgb(0, 0, 0), 1))  [static]
```

Gradient from white (0) to black (1).

Definition at line 23 of file Gradients.cs.

The documentation for this class was generated from the following file:

- VectSharp/Gradients.cs

7.63 VectSharp.GradientStop Struct Reference

Represents a colour stop in a gradient.

Public Member Functions

- GradientStop (Colour colour, double offset)

  Creates a new GradientStop instance.
- GradientStop MultiplyOpacity (double opacity)

  Returns a GradientStop corresponding to the current instance, whose colour's opacity has been multiplied by the specified value.

Properties

- Colour Colour [get]

  The Colour at the gradient stop.
- double Offset [get]

  The offset of the gradient stop. Range: [0, 1].

7.63.1 Detailed Description

Represents a colour stop in a gradient.

Definition at line 113 of file Brush.cs.

7.63.2 Constructor & Destructor Documentation

7.63.2.1 GradientStop()

```
VectSharp.GradientStop.GradientStop (
            Colour colour,
            double offset )
```

Creates a new GradientStop instance.

Parameters

colour	The Colour at the gradient stop.
offset	The offset of the gradient stop. Range: [0, 1].

Definition at line 130 of file Brush.cs.

7.63.3 Member Function Documentation

7.63.3.1 MultiplyOpacity()

```
GradientStop VectSharp.GradientStop.MultiplyOpacity (
            double opacity )
```

Returns a GradientStop corresponding to the current instance, whose colour's opacity has been multiplied by the specified value.

Parameters

opacity	The value that will be used to multiply the colour's opacity.

Returns

A GradientStop corresponding to the current instance, whose colour's opacity has been multiplied by the specified value.

Definition at line 142 of file Brush.cs.

7.63.4 Property Documentation

7.63.4.1 Colour

```
Colour VectSharp.GradientStop.Colour  [get]
```

The Colour at the gradient stop.

Definition at line 118 of file Brush.cs.

7.63.4.2 Offset

```
double VectSharp.GradientStop.Offset  [get]
```

The offset of the gradient stop. Range: [0, 1].

Definition at line 123 of file Brush.cs.

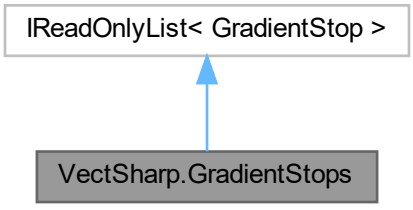
The documentation for this struct was generated from the following file:

- VectSharp/Brush.cs

7.64 VectSharp.GradientStops Class Reference

Represents a read-only list of GradientStops.

Inheritance diagram for VectSharp.GradientStops:



Public Member Functions

- IEnumerator< GradientStop > GetEnumerator ()
- GradientStops (IEnumerable< GradientStop > gradientStops)

  Creates a new GradientStops instance containing the specified gradient stops.
- GradientStops (params GradientStop[ ] gradientStops)

  Creates a new GradientStops instance containing the specified gradient stops.
- Colour GetColourAt (double position)

  Gets the colour at a certain position on the gradient.

Static Public Member Functions

- static implicit operator Func< double, Colour > (GradientStops stops)

  Convert a GradientStops object to a function that assigns a colour to values between 0 and 1.

Static Public Attributes

- static readonly double StopTolerance = 1e-7

  The minimum distance that is enforced between consecutive gradient stops.

Properties

- GradientStop this[int index] `[get]`
- int Count `[get]`

7.64.1 Detailed Description

Represents a read-only list of GradientStops.

Definition at line 152 of file Brush.cs.

7.64.2 Constructor & Destructor Documentation

7.64.2.1 GradientStops() `[1/2]`

```
VectSharp.GradientStops.GradientStops (
            IEnumerable< GradientStop > gradientStops )
```

Creates a new GradientStops instance containing the specified gradient stops.

Parameters

gradientStops	The gradient stops that will be contained in the GradientStops object.

Definition at line 182 of file Brush.cs.

7.64.2.2 GradientStops() [2/2]

```
VectSharp.GradientStops.GradientStops (
            params GradientStop[] gradientStops )
```

Creates a new GradientStops instance containing the specified gradient stops.

Parameters

gradientStops	The gradient stops that will be contained in the GradientStops object.

Definition at line 228 of file Brush.cs.

7.64.3 Member Function Documentation

7.64.3.1 GetColourAt()

```
Colour VectSharp.GradientStops.GetColourAt (
            double position )
```

Gets the colour at a certain position on the gradient.

Parameters

position	The position in the gradient (ranging from 0 to 1).

Returns

The colour of the gradient at the specified position.

Definition at line 238 of file Brush.cs.

7.64.3.2 GetEnumerator()

```
IEnumerator< GradientStop > VectSharp.GradientStops.GetEnumerator ( )
```

Definition at line 168 of file Brush.cs.

7.64.3.3 operator Func< double, Colour >()

```
static implicit VectSharp.GradientStops.operator Func< double, Colour > (
            GradientStops stops )  [static]
```

Convert a GradientStops object to a function that assigns a colour to values between 0 and 1.

Parameters

stops	The GradientStops to convert.

7.64.4 Member Data Documentation

7.64.4.1 StopTolerance

```
readonly double VectSharp.GradientStops.StopTolerance = 1e-7  [static]
```

The minimum distance that is enforced between consecutive gradient stops.

Definition at line 157 of file Brush.cs.

7.64.5 Property Documentation

7.64.5.1 Count

```
int VectSharp.GradientStops.Count  [get]
```

Definition at line 163 of file Brush.cs.

7.64.5.2 this[int index]

```
GradientStop VectSharp.GradientStops.this[int index]  [get]
```

Definition at line 160 of file Brush.cs.

The documentation for this class was generated from the following file:

- VectSharp/Brush.cs

7.65 VectSharp.Graphics Class Reference

Represents an abstract drawing surface.

Public Member Functions

- void FillPath (GraphicsPath path, Brush fillColour, string tag=null)

  Fill a GraphicsPath.
- void FillPath (GraphicsPath path, Brush fillColour, FillRule fillRule, string tag=null)

  Fill a GraphicsPath.
- void StrokePath (GraphicsPath path, Brush strokeColour, double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter, LineDash? lineDash=null, string tag=null)

  Stroke a GraphicsPath.
- void SetClippingPath (GraphicsPath path, string tag=null)

  Intersect the current clipping path with the specified GraphicsPath.
- void SetClippingPath (double leftX, double topY, double width, double height, string tag=null)

  Intersect the current clipping path with the specified rectangle.
- void SetClippingPath (Point topLeft, Size size, string tag=null)

  Intersect the current clipping path with the specified rectangle.
- void Rotate (double angle, string tag=null)

  Rotate the coordinate system around the origin.
- void RotateAt (double angle, Point pivot, string tag=null)

  Rotate the coordinate system around a pivot point.
- void Transform (double a, double b, double c, double d, double e, double f, string tag=null)

  Transform the coordinate system with the specified transformation matrix [[a, c, e], [b, d, f], [0, 0, 1]].
- void Translate (double x, double y, string tag=null)

  Translate the coordinate system origin.
- void Translate (Point delta, string tag=null)

  Translate the coordinate system origin.
- void Scale (double scaleX, double scaleY, string tag=null)

  Scale the coordinate system with respect to the origin.
- void FillRectangle (Point topLeft, Size size, Brush fillColour, string tag=null)

  Fill a rectangle.
- void FillRectangle (double leftX, double topY, double width, double height, Brush fillColour, string tag=null)

  Fill a rectangle.
- void StrokeRectangle (Point topLeft, Size size, Brush strokeColour, double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter, LineDash? lineDash=null, string tag=null)

  Stroke a rectangle.
- void StrokeRectangle (double leftX, double topY, double width, double height, Brush strokeColour, double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter, LineDash? lineDash=null, string tag=null)

  Stroke a rectangle.
- void DrawRasterImage (int sourceX, int sourceY, int sourceWidth, int sourceHeight, double destinationX, double destinationY, double destinationWidth, double destinationHeight, RasterImage image, string tag=null)

  Draw a raster image.
- void DrawRasterImage (double x, double y, RasterImage image, string tag=null)

  Draw a raster image.
- void DrawRasterImage (Point position, RasterImage image, string tag=null)

  Draw a raster image.
- void DrawRasterImage (double x, double y, double width, double height, RasterImage image, string tag=null)

Draw a raster image.

- void DrawRasterImage (Point position, Size size, RasterImage image, string tag=null)

  Draw a raster image.
- void Save ()

  Save the current transform state (rotation, translation, scale).
- void Restore ()

  Restore the previous transform state (rotation, translation scale).
- void CopyToIGraphicsContext (IGraphicsContext destinationContext)

  Copy the current graphics to an instance of a class implementing IGraphicsContext.
- void DrawGraphics (Point origin, Graphics graphics)

  Draws a Graphics object on the current Graphics object.
- void DrawGraphics (double originX, double originY, Graphics graphics, string tag)

  Draws a Graphics object on the current Graphics object, prepending the supplied tag to the tags contained in the Graphics object being drawn.
- void DrawGraphics (Point origin, Graphics graphics, string tag)

  Draws a Graphics object on the current Graphics object, prepending the supplied tag to the tags contained in the Graphics object being drawn.
- void DrawGraphics (double originX, double originY, Graphics graphics)

  Draws a Graphics object on the current Graphics object.
- void DrawGraphics (Point origin, Graphics graphics, IFilter filter, string tag=null)

  Draws a Graphics object on the current Graphics object, applying the specified filter .
- void DrawGraphics (double originX, double originY, Graphics graphics, IFilter filter, string tag=null)

  Draws a Graphics object on the current Graphics object, applying the specified filter .
- Graphics Transform (Func< Point, Point > transformationFunction, double linearisationResolution)

  Creates a new Graphics object in which all the graphics actions have been transformed using an arbitrary transformation function. Raster images are replaced by grey rectangles.
- Graphics Transform (Func< Point, Point > transformationFunction, double linearisationResolution, double maxSegmentLength)

  Creates a new Graphics object in which all the graphics actions have been transformed using an arbitrary transformation function. Raster images are replaced by grey rectangles.
- Graphics Linearise (double resolution)

  Creates a new Graphics object by linearising all of the elements of the current instance, i.e. replacing curve segments with series of line segments that approximate them. Raster images are left unchanged.
- Rectangle GetBounds ()

  Computes the rectangular bounds of the region affected by the drawing operations performed on the Graphics object.
- bool TryRasterise (Rectangle region, double scale, bool interpolate, out RasterImage output)

  Tries to rasterise specified region of this Graphics object using the default rasterisation method.
- void Crop (Rectangle region)

  Removes graphics actions that fall completely outside of the specified region .
- void Crop (Point topLeft, Size size)

  Removes graphics actions that fall completely outside of the specified region.
- IEnumerable< string > GetTags ()

  Gets all the tags that have been defined in the Graphics.
- void FillText (Point origin, string text, Font font, Brush fillColour, TextBaselines textBaseline=TextBaselines.↩ Top, string tag=null)

  Fill a text string.
- void FillText (Point origin, string text, Font font, Brush fillColour, TextSpacing spacing, TextBaselines text↩ Baseline=TextBaselines.Top, string tag=null)

  Fill a text string.
- void FillText (double originX, double originY, string text, Font font, Brush fillColour, TextBaselines text↩ Baseline=TextBaselines.Top, string tag=null)

  Fill a text string.

- void FillText (double originX, double originY, string text, Font font, Brush fillColour, TextSpacing spacing, TextBaselines textBaseline=TextBaselines.Top, string tag=null)

  Fill a text string.
- void StrokeText (Point origin, string text, Font font, Brush strokeColour, TextBaselines textBaseline=TextBaselines.Top, double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter, LineDash? lineDash=null, string tag=null)

  Stroke a text string.
- void StrokeText (Point origin, string text, Font font, Brush strokeColour, TextSpacing spacing, TextBaselines textBaseline=TextBaselines.Top, double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter, LineDash? lineDash=null, string tag=null)

  Stroke a text string.
- void StrokeText (double originX, double originY, string text, Font font, Brush strokeColour, TextBaselines textBaseline=TextBaselines.Top, double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter, LineDash? lineDash=null, string tag=null)

  Stroke a text string.
- void StrokeText (double originX, double originY, string text, Font font, Brush strokeColour, TextSpacing spacing, TextBaselines textBaseline=TextBaselines.Top, double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter, LineDash? lineDash=null, string tag=null)

  Stroke a text string.
- void FillTextOnPath (GraphicsPath path, string text, Font font, Brush fillColour, double reference=0, TextAnchors anchor=TextAnchors.Left, TextBaselines textBaseline=TextBaselines.Top, string tag=null)

  Fill a text string along a GraphicsPath.
- void StrokeTextOnPath (GraphicsPath path, string text, Font font, Brush strokeColour, double reference=0, TextAnchors anchor=TextAnchors.Left, TextBaselines textBaseline=TextBaselines.Top, double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter, LineDash? lineDash=null, string tag=null)

  Stroke a text string along a GraphicsPath.
- void FillText (Point origin, IEnumerable< FormattedText > text, Brush fillColour, TextBaselines textBaseline=TextBaselines.Top, string tag=null)

  Fill a formatted text string.
- void FillText (double originX, double originY, IEnumerable< FormattedText > text, Brush fillColour, TextBaselines textBaseline=TextBaselines.Top, string tag=null)

  Fill a formatted text string.
- void StrokeText (Point origin, IEnumerable< FormattedText > text, Brush strokeColour, TextBaselines textBaseline=TextBaselines.Top, double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter, LineDash? lineDash=null, string tag=null)

  Stroke a formatted text string.
- void StrokeText (double originX, double originY, IEnumerable< FormattedText > text, Brush strokeColour, TextBaselines textBaseline=TextBaselines.Top, double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter, LineDash? lineDash=null, string tag=null)

  Stroke a formatted text string.
- Size MeasureText (string text, Font font)

  Measure a text string. See also

  See also

  Font.MeasureText(string), Font.MeasureTextAdvanced(string)

  and .
- Size MeasureText (IEnumerable< FormattedText > text)

  Measure a formatted text string. See also

See also

FormattedTextExtensions.Measure(IEnumerable<FormattedText>)

.

- void FillTextUnderline (double originX, double originY, string text, Font font, Brush fillColour, TextBaselines textBaseline=TextBaselines.Top, string tag=null)

  Fills the underline of the specified text string.
- void FillTextUnderline (Point origin, string text, Font font, Brush fillColour, TextBaselines textBaseline=TextBaselines.Top, string tag=null)

  Fills the underline of the specified text string.
- void StrokeTextUnderline (double originX, double originY, string text, Font font, Brush strokeColour, TextBaselines textBaseline=TextBaselines.Top, double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter, LineDash? lineDash=null, string tag=null)

  Stroke the underline of the specified text string.
- void StrokeTextUnderline (Point origin, string text, Font font, Brush strokeColour, TextBaselines textBaseline=TextBaselines.Top, double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter, LineDash? lineDash=null, string tag=null)

  Stroke the underline of the specified text string.
- void FillTextUnderline (double originX, double originY, IEnumerable< FormattedText > text, Brush fillColour, TextBaselines textBaseline=TextBaselines.Top, string tag=null)

  Fill the underline of the specified formatted text string.
- void FillTextUnderline (Point origin, IEnumerable< FormattedText > text, Brush fillColour, TextBaselines textBaseline=TextBaselines.Top, string tag=null)

  Fill the underline of the specified formatted text string.
- void StrokeTextUnderline (double originX, double originY, IEnumerable< FormattedText > text, Brush strokeColour, TextBaselines textBaseline=TextBaselines.Top, double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter, LineDash? lineDash=null, string tag=null)

  Stroke the underline of the specified formatted text string.
- void StrokeTextUnderline (Point origin, IEnumerable< FormattedText > text, Brush strokeColour, TextBaselines textBaseline=TextBaselines.Top, double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter, LineDash? lineDash=null, string tag=null)

  Stroke the underline of the specified formatted text string.

Static Public Attributes

- static Func< Graphics, Rectangle, double, bool, RasterImage > RasterisationMethod = null

  A method that is used to rasterise a region of a Graphics object. Set this to `null` if you wish to use the default rasterisation methods (implemented by either VectSharp.Raster, or VectSharp.Raster.ImageSharp). You will have to provide your own implementation of this method if neither VectSharp.Raster nor VectSharp.Raster.ImageSharp are referenced by your project. The first argument of this method is the Graphics to be rasterised, the second is a Rectangle representing the region to rasterise, the third is a double representing the scale, and the third is a boolean value indicating whether the resulting RasterImage should be interpolated.

Properties

- static UnbalancedStackActions UnbalancedStackAction = UnbalancedStackActions.Throw `[get, set]`

  Determines how an unbalanced graphics state stack (which occurs if the number of calls to Save and Restore is not equal) will be treated. The default is UnbalancedStackActions.Throw.
- FillRule DefaultFillRule = FillRule.NonZeroWinding `[get, set]`

  The default fill rule.
- bool UseUniqueTags = true `[get, set]`

  Determines whether unique tags should be used for graphics actions that create multiple objects (e.g. drawing text).

7.65.1 Detailed Description

Represents an abstract drawing surface.

Definition at line 279 of file Graphics.cs.

7.65.2 Member Function Documentation

7.65.2.1 CopyToIGraphicsContext()

```
void VectSharp.Graphics.CopyToIGraphicsContext (
            IGraphicsContext destinationContext )
```

Copy the current graphics to an instance of a class implementing IGraphicsContext.

Parameters

destinationContext	The IGraphicsContext on which the graphics are to be copied.

Definition at line 667 of file Graphics.cs.

7.65.2.2 Crop() [1/2]

```
void VectSharp.Graphics.Crop (
            Point topLeft,
            Size size )
```

Removes graphics actions that fall completely outside of the specified region.

Parameters

topLeft	The top-left corner of the area to preserve.
size	The size of the area to preserve.

Definition at line 1700 of file Graphics.cs.

7.65.2.3 Crop() [2/2]

```
void VectSharp.Graphics.Crop (
            Rectangle region )
```

Removes graphics actions that fall completely outside of the specified *region* .

Parameters

region	The area to preserve.

Definition at line 1611 of file Graphics.cs.

7.65.2.4 DrawGraphics() [1/6]

```
void VectSharp.Graphics.DrawGraphics (
            double originX,
            double originY,
            Graphics graphics )
```

Draws a Graphics object on the current Graphics object.

Parameters

originX	The horizontal coordinate at which to place the origin of *graphics* .
originY	The vertical coordinate at which to place the origin of *graphics* .
graphics	The Graphics object to draw on the current Graphics object.

Definition at line 951 of file Graphics.cs.

7.65.2.5 DrawGraphics() [2/6]

```
void VectSharp.Graphics.DrawGraphics (
            double originX,
            double originY,
            Graphics graphics,
            IFilter filter,
            string tag = null )
```

Draws a Graphics object on the current Graphics object, applying the specified *filter* .

Parameters

originX	The horizontal coordinate at which to place the origin of *graphics* .
originY	The vertical coordinate at which to place the origin of *graphics* .
graphics	The Graphics object to draw on the current Graphics object.
filter	An IFilter object, representing the filter to apply to the *graphics* object.
tag	A tag to identify the filter.

Definition at line 993 of file Graphics.cs.

7.65.2.6 DrawGraphics() [3/6]

```
void VectSharp.Graphics.DrawGraphics (
            double originX,
            double originY,
            Graphics graphics,
            string tag )
```

Draws a Graphics object on the current Graphics object, prepending the supplied *tag* to the tags contained in the Graphics object being drawn.

Parameters

originX	The horizontal coordinate at which to place the origin of *graphics* .
originY	The vertical coordinate at which to place the origin of *graphics* .
graphics	The Graphics object to draw on the current Graphics object.
tag	The tag to prepend to the tags contained in the *graphics* object.

Definition at line 894 of file Graphics.cs.

7.65.2.7 DrawGraphics() [4/6]

```
void VectSharp.Graphics.DrawGraphics (
            Point origin,
            Graphics graphics )
```

Draws a Graphics object on the current Graphics object.

Parameters

origin	The point at which to place the origin of *graphics* .
graphics	The Graphics object to draw on the current Graphics object.

Definition at line 875 of file Graphics.cs.

7.65.2.8 DrawGraphics() [5/6]

```
void VectSharp.Graphics.DrawGraphics (
            Point origin,
            Graphics graphics,
            IFilter filter,
            string tag = null )
```

Draws a Graphics object on the current Graphics object, applying the specified *filter* .

Parameters

origin	The point at which to place the origin of *graphics* .
graphics	The Graphics object to draw on the current Graphics object.
filter	An IFilter object, representing the filter to apply to the *graphics* object.
tag	A tag to identify the filter.

Definition at line 963 of file Graphics.cs.

7.65.2.9 DrawGraphics() [6/6]

```
void VectSharp.Graphics.DrawGraphics (
            Point origin,
            Graphics graphics,
            string tag )
```

Draws a Graphics object on the current Graphics object, prepending the supplied *tag* to the tags contained in the Graphics object being drawn.

Parameters

origin	The point at which to place the origin of *graphics* .
graphics	The Graphics object to draw on the current Graphics object.
tag	The tag to prepend to the tags contained in the *graphics* object.

Definition at line 905 of file Graphics.cs.

7.65.2.10 DrawRasterImage() [1/5]

```
void VectSharp.Graphics.DrawRasterImage (
            double x,
            double y,
            double width,
            double height,
            RasterImage image,
            string tag = null )
```

Draw a raster image.

Parameters

x	The horizontal coordinate of the top-left corner of the rectangle delimiting the destination area of the image.
y	The vertical coordinate of the top-left corner of the rectangle delimiting the destination area of the image.
width	The width of the rectangle delimiting the destination area of the image.
height	The height of the rectangle delimiting the destination area of the image.
image	The image to draw.
tag	A tag to identify the drawn image.

Definition at line 572 of file Graphics.cs.

7.65.2.11 DrawRasterImage() [2/5]

```
void VectSharp.Graphics.DrawRasterImage (
            double x,
            double y,
            RasterImage image,
            string tag = null )
```

Draw a raster image.

Parameters

x	The horizontal coordinate of the top-left corner of the rectangle delimiting the destination area of the image.
y	The vertical coordinate of the top-left corner of the rectangle delimiting the destination area of the image.
image	The image to draw.
tag	A tag to identify the drawn image.

Definition at line 547 of file Graphics.cs.

7.65.2.12 DrawRasterImage() [3/5]

```
void VectSharp.Graphics.DrawRasterImage (
            int sourceX,
            int sourceY,
            int sourceWidth,
            int sourceHeight,
            double destinationX,
            double destinationY,
            double destinationWidth,
            double destinationHeight,
            RasterImage image,
            string tag = null )
```

Draw a raster image.

Parameters

sourceX	The horizontal coordinate of the top-left corner of the rectangle delimiting the source area of the image.
sourceY	The vertical coordinate of the top-left corner of the rectangle delimiting the source area of the image.
sourceWidth	The width of the rectangle delimiting the source area of the image.
sourceHeight	The height of the rectangle delimiting the source area of the image.

Parameters

destinationX	The horizontal coordinate of the top-left corner of the rectangle delimiting the destination area of the image.
destinationY	The vertical coordinate of the top-left corner of the rectangle delimiting the destination area of the image.
destinationWidth	The width of the rectangle delimiting the destination area of the image.
destinationHeight	The height of the rectangle delimiting the destination area of the image.
image	The image to draw.
tag	A tag to identify the drawn image.

Definition at line 535 of file Graphics.cs.

7.65.2.13 DrawRasterImage() [4/5]

```
void VectSharp.Graphics.DrawRasterImage (
            Point position,
            RasterImage image,
            string tag = null )
```

Draw a raster image.

Parameters

position	The the top-left corner of the rectangle delimiting the destination area of the image.
image	The image to draw.
tag	A tag to identify the drawn image.

Definition at line 558 of file Graphics.cs.

7.65.2.14 DrawRasterImage() [5/5]

```
void VectSharp.Graphics.DrawRasterImage (
            Point position,
            Size size,
            RasterImage image,
            string tag = null )
```

Draw a raster image.

Parameters

position	The the top-left corner of the rectangle delimiting the destination area of the image.
size	The size of the rectangle delimiting the destination area of the image.
image	The image to draw.
tag	A tag to identify the drawn image.

Definition at line 584 of file Graphics.cs.

7.65.2.15 FillPath() [1/2]

```
void VectSharp.Graphics.FillPath (
            GraphicsPath path,
            Brush fillColour,
            FillRule fillRule,
            string tag = null )
```

Fill a GraphicsPath.

Parameters

path	The GraphicsPath to fill.
fillColour	The Brush with which to fill the GraphicsPath.
fillRule	The FillRule that determines which parts of the path are filled.
tag	A tag to identify the filled path.

Definition at line 311 of file Graphics.cs.

7.65.2.16 FillPath() [2/2]

```
void VectSharp.Graphics.FillPath (
            GraphicsPath path,
            Brush fillColour,
            string tag = null )
```

Fill a GraphicsPath.

Parameters

path	The GraphicsPath to fill.
fillColour	The Brush with which to fill the GraphicsPath.
tag	A tag to identify the filled path.

Definition at line 299 of file Graphics.cs.

7.65.2.17 FillRectangle() [1/2]

```
void VectSharp.Graphics.FillRectangle (
            double leftX,
```

```
            double topY,
            double width,
            double height,
            Brush fillColour,
            string tag = null )
```

Fill a rectangle.

Parameters

leftX	The horizontal coordinate of the top-left corner of the rectangle.
topY	The vertical coordinate of the top-left corner of the rectangle.
width	The width of the rectangle.
height	The height of the rectangle.
fillColour	The colour with which to fill the rectangle.
tag	A tag to identify the filled rectangle.

Definition at line 483 of file Graphics.cs.

7.65.2.18 FillRectangle() [2/2]

```
void VectSharp.Graphics.FillRectangle (
            Point topLeft,
            Size size,
            Brush fillColour,
            string tag = null )
```

Fill a rectangle.

Parameters

topLeft	The top-left corner of the rectangle.
size	The size of the rectangle.
fillColour	The colour with which to fill the rectangle.
tag	A tag to identify the filled rectangle.

Definition at line 469 of file Graphics.cs.

7.65.2.19 FillText() [1/6]

```
void VectSharp.Graphics.FillText (
            double originX,
            double originY,
            IEnumerable< FormattedText > text,
            Brush fillColour,
```

```
            TextBaselines textBaseline = TextBaselines.Top,
            string tag = null )
```

Fill a formatted text string.

Parameters

originX	The horizontal coordinate of the text origin.
originY	The vertical coordinate of the text origin. See *textBaseline* .
text	The FormattedText to draw.
fillColour	The default Brush to use to fill the text. This can be overridden by each *text* element.
textBaseline	The text baseline (determines what *originY* represents).
tag	A tag to identify the filled text.

Definition at line 638 of file Graphics.Text.cs.

7.65.2.20 FillText() [2/6]

```
void VectSharp.Graphics.FillText (
            double originX,
            double originY,
            string text,
            Font font,
            Brush fillColour,
            TextBaselines textBaseline = TextBaselines.Top,
            string tag = null )
```

Fill a text string.

Parameters

originX	The horizontal coordinate of the text origin.
originY	The vertical coordinate of the text origin. See *textBaseline* .
text	The string to draw.
font	The font with which to draw the text.
fillColour	The Brush to use to fill the text.
textBaseline	The text baseline (determines what *originY* represents).
tag	A tag to identify the filled text.

Definition at line 79 of file Graphics.Text.cs.

7.65.2.21 FillText() [3/6]

```
void VectSharp.Graphics.FillText (
            double originX,
            double originY,
            string text,
            Font font,
            Brush fillColour,
            TextSpacing spacing,
```

```
            TextBaselines textBaseline = TextBaselines.Top,
            string tag = null )
```

Fill a text string.

Parameters

originX	The horizontal coordinate of the text origin.
originY	The vertical coordinate of the text origin. See *textBaseline* .
text	The string to draw.
font	The font with which to draw the text.
fillColour	The Brush to use to fill the text.
spacing	Spacing parameters to use for the text.
textBaseline	The text baseline (determines what *originY* represents).
tag	A tag to identify the filled text.

Definition at line 103 of file Graphics.Text.cs.

7.65.2.22 FillText() [4/6]

```
void VectSharp.Graphics.FillText (
            Point origin,
            IEnumerable< FormattedText > text,
            Brush fillColour,
            TextBaselines textBaseline = TextBaselines.Top,
            string tag = null )
```

Fill a formatted text string.

Parameters

origin	The text origin. See *textBaseline* .
text	The FormattedText to draw.
fillColour	The default Brush to use to fill the text. This can be overridden by each *text* element.
textBaseline	The text baseline (determines what the vertical component of *origin* represents).
tag	A tag to identify the filled text.

Definition at line 509 of file Graphics.Text.cs.

7.65.2.23 FillText() [5/6]

```
void VectSharp.Graphics.FillText (
            Point origin,
            string text,
            Font font,
            Brush fillColour,
            TextBaselines textBaseline = TextBaselines.Top,
            string tag = null )
```

Fill a text string.

Parameters

origin	The text origin. See *textBaseline* .
text	The string to draw.
font	The font with which to draw the text.
fillColour	The Brush to use to fill the text.
textBaseline	The text baseline (determines what the vertical component of *origin* represents).
tag	A tag to identify the filled text.

Definition at line 34 of file Graphics.Text.cs.

7.65.2.24 FillText() [6/6]

```
void VectSharp.Graphics.FillText (
            Point origin,
            string text,
            Font font,
            Brush fillColour,
            TextSpacing spacing,
            TextBaselines textBaseline = TextBaselines.Top,
            string tag = null )
```

Fill a text string.

Parameters

origin	The text origin. See *textBaseline* .
text	The string to draw.
font	The font with which to draw the text.
fillColour	The Brush to use to fill the text.
spacing	Spacing parameters to use for the text.
textBaseline	The text baseline (determines what the vertical component of *origin* represents).
tag	A tag to identify the filled text.

Definition at line 57 of file Graphics.Text.cs.

7.65.2.25 FillTextOnPath()

```
void VectSharp.Graphics.FillTextOnPath (
            GraphicsPath path,
            string text,
            Font font,
            Brush fillColour,
            double reference = 0,
            TextAnchors anchor = TextAnchors.Left,
```

```
TextBaselines textBaseline = TextBaselines.Top,
string tag = null )
```

Fill a text string along a GraphicsPath.

Parameters

path	The GraphicsPath along which the text will flow.
text	The string to draw.
font	The font with which to draw the text.
fillColour	The Brush to use to fill the text.
reference	The (relative) starting point on the path starting from which the text should be drawn (0 is the start of the path, 1 is the end of the path).
anchor	The anchor in the text string that will correspond to the point specified by the *reference* .
textBaseline	The text baseline (determines which the position of the text in relation to the *path* .
tag	A tag to identify the filled text.

Definition at line 232 of file Graphics.Text.cs.

7.65.2.26 FillTextUnderline() [1/4]

```
void VectSharp.Graphics.FillTextUnderline (
            double originX,
            double originY,
            IEnumerable< FormattedText > text,
            Brush fillColour,
            TextBaselines textBaseline = TextBaselines.Top,
            string tag = null )
```

Fill the underline of the specified formatted text string.

Parameters

originX	The horizontal coordinate of the text origin.
originY	The vertical coordinate of the text origin. See *textBaseline* .
text	The FormattedText whose underline will be drawn.
fillColour	The default Brush to use to fill the underline. This can be overridden by each *text* element.
textBaseline	The text baseline (determines what *originY* represents).
tag	A tag to identify the filled underlined.

Definition at line 908 of file Graphics.Text.cs.

7.65.2.27 FillTextUnderline() [2/4]

```
void VectSharp.Graphics.FillTextUnderline (
            double originX,
            double originY,
            string text,
            Font font,
            Brush fillColour,
```

```
            TextBaselines textBaseline = TextBaselines.Top,
            string tag = null )
```

Fills the underline of the specified text string.

Parameters

originX	The horizontal coordinate of the text origin.
originY	The vertical coordinate of the text origin. See *textBaseline* .
text	The string whose underline will be draw.
font	The font with which to draw the text.
fillColour	The Brush to use to fill the underline.
textBaseline	The text baseline (determines what *originY* represents).
tag	A tag to identify the filled underline.

Definition at line 834 of file Graphics.Text.cs.

7.65.2.28 FillTextUnderline() [3/4]

```
void VectSharp.Graphics.FillTextUnderline (
            Point origin,
            IEnumerable< FormattedText > text,
            Brush fillColour,
            TextBaselines textBaseline = TextBaselines.Top,
            string tag = null )
```

Fill the underline of the specified formatted text string.

Parameters

origin	The text origin. See *textBaseline* .
text	The FormattedText whose underline will be drawn.
fillColour	The default Brush to use to fill the underline. This can be overridden by each *text* element.
textBaseline	The text baseline (determines what the vertical component of *origin* represents).
tag	A tag to identify the filled underlined.

Definition at line 921 of file Graphics.Text.cs.

7.65.2.29 FillTextUnderline() [4/4]

```
void VectSharp.Graphics.FillTextUnderline (
            Point origin,
            string text,
            Font font,
            Brush fillColour,
            TextBaselines textBaseline = TextBaselines.Top,
            string tag = null )
```

Fills the underline of the specified text string.

Parameters

origin	The text origin. See *textBaseline* .
text	The string whose underline will be draw.
font	The font with which to draw the text.
fillColour	The Brush to use to fill the underline.
textBaseline	The text baseline (determines what the vertical component of *origin* represents).
tag	A tag to identify the filled underline.

Definition at line 848 of file Graphics.Text.cs.

7.65.2.30 GetBounds()

```
Rectangle VectSharp.Graphics.GetBounds ( )
```

Computes the rectangular bounds of the region affected by the drawing operations performed on the Graphics object.

Returns

The smallest rectangle that contains all the elements drawn on the Graphics.

Definition at line 1458 of file Graphics.cs.

7.65.2.31 GetTags()

```
IEnumerable< string > VectSharp.Graphics.GetTags ( )
```

Gets all the tags that have been defined in the Graphics.

Returns

An IEnumerable<T> of strings that, when enumerated, returns all the tags that have been defined in the Graphics.

Definition at line 1710 of file Graphics.cs.

7.65.2.32 Linearise()

```
Graphics VectSharp.Graphics.Linearise (
            double resolution )
```

Creates a new Graphics object by linearising all of the elements of the current instance, i.e. replacing curve segments with series of line segments that approximate them. Raster images are left unchanged.

Parameters

resolution	The resolution that will be used to linearise curve segments.

Returns

A new Graphics object containing the linearised elements.

Definition at line 1401 of file Graphics.cs.

7.65.2.33 MeasureText() [1/2]

```
Size VectSharp.Graphics.MeasureText (
            IEnumerable< FormattedText > text )
```

Measure a formatted text string. See also

See also

FormattedTextExtensions.Measure(IEnumerable<FormattedText>)

.

Parameters

text	The collection of FormattedText objects to measure.

Returns

The size of the measured *text* .

Definition at line 817 of file Graphics.Text.cs.

7.65.2.34 MeasureText() [2/2]

```
Size VectSharp.Graphics.MeasureText (
            string text,
            Font font )
```

Measure a text string. See also

See also

Font.MeasureText(string), Font.MeasureTextAdvanced(string)

and .

Parameters

text	The string to measure.
font	The font to use to measure the string.

Returns

The size of the measured *text* .

Definition at line 799 of file Graphics.Text.cs.

7.65.2.35 Restore()

```
void VectSharp.Graphics.Restore ( )
```

Restore the previous transform state (rotation, translation scale).

Definition at line 600 of file Graphics.cs.

7.65.2.36 Rotate()

```
void VectSharp.Graphics.Rotate (
            double angle,
            string tag = null )
```

Rotate the coordinate system around the origin.

Parameters

angle	The angle (in radians) by which to rotate the coordinate system.
tag	A tag to identify the transform.

Definition at line 375 of file Graphics.cs.

7.65.2.37 RotateAt()

```
void VectSharp.Graphics.RotateAt (
            double angle,
            Point pivot,
            string tag = null )
```

Rotate the coordinate system around a pivot point.

Parameters

angle	The angle (in radians) by which to rotate the coordinate system.
pivot	The pivot around which the coordinate system is to be rotated.
tag	A tag to identify the transform.

Definition at line 386 of file Graphics.cs.

7.65.2.38 Save()

```
void VectSharp.Graphics.Save ( )
```

Save the current transform state (rotation, translation, scale).

Definition at line 592 of file Graphics.cs.

7.65.2.39 Scale()

```
void VectSharp.Graphics.Scale (
            double scaleX,
            double scaleY,
            string tag = null )
```

Scale the coordinate system with respect to the origin.

Parameters

scaleX	The horizontal scale.
scaleY	The vertical scale.
tag	A tag to identify the transform.

Definition at line 457 of file Graphics.cs.

7.65.2.40 SetClippingPath() [1/3]

```
void VectSharp.Graphics.SetClippingPath (
            double leftX,
            double topY,
            double width,
            double height,
            string tag = null )
```

Intersect the current clipping path with the specified rectangle.

Parameters

leftX	The horizontal coordinate of the top-left corner of the rectangle.
topY	The vertical coordinate of the top-left corner of the rectangle.
width	The width of the rectangle.
height	The height of the rectangle.
tag	A tag to identify the clipping path.

Definition at line 354 of file Graphics.cs.

7.65.2.41 SetClippingPath() [2/3]

```
void VectSharp.Graphics.SetClippingPath (
            GraphicsPath path,
            string tag = null )
```

Intersect the current clipping path with the specified GraphicsPath.

Parameters

path	The GraphicsPath to intersect with the current clipping path.
tag	A tag to identify the clipping path.

Definition at line 341 of file Graphics.cs.

7.65.2.42 SetClippingPath() [3/3]

```
void VectSharp.Graphics.SetClippingPath (
            Point topLeft,
            Size size,
            string tag = null )
```

Intersect the current clipping path with the specified rectangle.

Parameters

topLeft	The top-left corner of the rectangle.
size	The size of the rectangle.
tag	A tag to identify the clipping path.

Definition at line 365 of file Graphics.cs.

7.65.2.43 StrokePath()

```
void VectSharp.Graphics.StrokePath (
            GraphicsPath path,
            Brush strokeColour,
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter,
            LineDash?  lineDash = null,
            string tag = null )
```

Stroke a GraphicsPath.

Parameters

path	The GraphicsPath to stroke.
strokeColour	The Brush with which to stroke the GraphicsPath.
lineWidth	The width of the line with which the path is stroked.
lineCap	The line cap to use to stroke the path.
lineJoin	The line join to use to stroke the path.
lineDash	The line dash to use to stroke the path.
tag	A tag to identify the stroked path.

Definition at line 331 of file Graphics.cs.

7.65.2.44 StrokeRectangle() [1/2]

```
void VectSharp.Graphics.StrokeRectangle (
            double leftX,
            double topY,
            double width,
            double height,
            Brush strokeColour,
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter,
            LineDash?  lineDash = null,
            string tag = null )
```

Stroke a rectangle.

Parameters

leftX	The horizontal coordinate of the top-left corner of the rectangle.
topY	The vertical coordinate of the top-left corner of the rectangle.
width	The width of the rectangle.
height	The height of the rectangle.
strokeColour	The colour with which to stroke the rectangle.
lineWidth	The width of the line with which the rectangle is stroked.
lineCap	The line cap to use to stroke the rectangle.
lineJoin	The line join to use to stroke the rectangle.
lineDash	The line dash to use to stroke the rectangle.
tag	A tag to identify the filled rectangle.

Definition at line 517 of file Graphics.cs.

7.65.2.45 StrokeRectangle() [2/2]

```
void VectSharp.Graphics.StrokeRectangle (
            Point topLeft,
            Size size,
            Brush strokeColour,
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter,
            LineDash?  lineDash = null,
            string tag = null )
```

Stroke a rectangle.

Parameters

topLeft	The top-left corner of the rectangle.
size	The size of the rectangle.
strokeColour	The colour with which to stroke the rectangle.
lineWidth	The width of the line with which the rectangle is stroked.
lineCap	The line cap to use to stroke the rectangle.
lineJoin	The line join to use to stroke the rectangle.
lineDash	The line dash to use to stroke the rectangle.
tag	A tag to identify the filled rectangle.

Definition at line 499 of file Graphics.cs.

7.65.2.46 StrokeText() [1/6]

```
void VectSharp.Graphics.StrokeText (
            double originX,
            double originY,
            IEnumerable< FormattedText > text,
            Brush strokeColour,
            TextBaselines textBaseline = TextBaselines.Top,
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter,
            LineDash?  lineDash = null,
            string tag = null )
```

Stroke a formatted text string.

Parameters

originX	The horizontal coordinate of the text origin.

Parameters

originY	The vertical coordinate of the text origin. See *textBaseline* .
text	The FormattedText to draw.
strokeColour	The default Brush with which to stroke the text.
lineWidth	The width of the line with which the text is stroked.
lineCap	The line cap to use to stroke the text.
lineJoin	The line join to use to stroke the text.
lineDash	The line dash to use to stroke the text.
textBaseline	The text baseline (determines what *originY* represents).
tag	A tag to identify the stroked text.

Definition at line 787 of file Graphics.Text.cs.

7.65.2.47 StrokeText() [2/6]

```
void VectSharp.Graphics.StrokeText (
            double originX,
            double originY,
            string text,
            Font font,
            Brush strokeColour,
            TextBaselines textBaseline = TextBaselines.Top,
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter,
            LineDash?  lineDash = null,
            string tag = null )
```

Stroke a text string.

Parameters

originX	The horizontal coordinate of the text origin.
originY	The vertical coordinate of the text origin. See *textBaseline* .
text	The string to draw.
font	The font with which to draw the text.
strokeColour	The Brush with which to stroke the text.
lineWidth	The width of the line with which the text is stroked.
lineCap	The line cap to use to stroke the text.
lineJoin	The line join to use to stroke the text.
lineDash	The line dash to use to stroke the text.
textBaseline	The text baseline (determines what *originY* represents).
tag	A tag to identify the stroked text.

Definition at line 181 of file Graphics.Text.cs.

7.65.2.48 StrokeText() [3/6]

```
void VectSharp.Graphics.StrokeText (
            double originX,
            double originY,
            string text,
            Font font,
            Brush strokeColour,
            TextSpacing spacing,
            TextBaselines textBaseline = TextBaselines.Top,
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter,
            LineDash?  lineDash = null,
            string tag = null )
```

Stroke a text string.

Parameters

originX	The horizontal coordinate of the text origin.
originY	The vertical coordinate of the text origin. See *textBaseline* .
text	The string to draw.
font	The font with which to draw the text.
strokeColour	The Brush with which to stroke the text.
spacing	Spacing parameters to use for the text.
lineWidth	The width of the line with which the text is stroked.
lineCap	The line cap to use to stroke the text.
lineJoin	The line join to use to stroke the text.
lineDash	The line dash to use to stroke the text.
textBaseline	The text baseline (determines what *originY* represents).
tag	A tag to identify the stroked text.

Definition at line 209 of file Graphics.Text.cs.

7.65.2.49 StrokeText() [4/6]

```
void VectSharp.Graphics.StrokeText (
            Point origin,
            IEnumerable< FormattedText > text,
            Brush strokeColour,
            TextBaselines textBaseline = TextBaselines.Top,
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter,
            LineDash?  lineDash = null,
            string tag = null )
```

Stroke a formatted text string.

Parameters

origin	The text origin. See *textBaseline* .
text	The FormattedText to draw.
strokeColour	The default Brush with which to stroke the text.
lineWidth	The width of the line with which the text is stroked.
lineCap	The line cap to use to stroke the text.
lineJoin	The line join to use to stroke the text.
lineDash	The line dash to use to stroke the text.
textBaseline	The text baseline (determines what the vertical component of *origin* represents).
tag	A tag to identify the stroked text.

Definition at line 655 of file Graphics.Text.cs.

7.65.2.50 StrokeText() [5/6]

```
void VectSharp.Graphics.StrokeText (
            Point origin,
            string text,
            Font font,
            Brush strokeColour,
            TextBaselines textBaseline = TextBaselines.Top,
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter,
            LineDash?  lineDash = null,
            string tag = null )
```

Stroke a text string.

Parameters

origin	The text origin. See *textBaseline* .
text	The string to draw.
font	The font with which to draw the text.
strokeColour	The Brush with which to stroke the text.
lineWidth	The width of the line with which the text is stroked.
lineCap	The line cap to use to stroke the text.
lineJoin	The line join to use to stroke the text.
lineDash	The line dash to use to stroke the text.
textBaseline	The text baseline (determines what the vertical component of *origin* represents).
tag	A tag to identify the stroked text.

Definition at line 128 of file Graphics.Text.cs.

7.65.2.51 StrokeText() [6/6]

```
void VectSharp.Graphics.StrokeText (
            Point origin,
            string text,
            Font font,
            Brush strokeColour,
            TextSpacing spacing,
            TextBaselines textBaseline = TextBaselines.Top,
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter,
            LineDash?  lineDash = null,
            string tag = null )
```

Stroke a text string.

Parameters

origin	The text origin. See *textBaseline* .
text	The string to draw.
font	The font with which to draw the text.
strokeColour	The Brush with which to stroke the text.
spacing	Spacing parameters to use for the text.
lineWidth	The width of the line with which the text is stroked.
lineCap	The line cap to use to stroke the text.
lineJoin	The line join to use to stroke the text.
lineDash	The line dash to use to stroke the text.
textBaseline	The text baseline (determines what the vertical component of *origin* represents).
tag	A tag to identify the stroked text.

Definition at line 155 of file Graphics.Text.cs.

7.65.2.52 StrokeTextOnPath()

```
void VectSharp.Graphics.StrokeTextOnPath (
            GraphicsPath path,
            string text,
            Font font,
            Brush strokeColour,
            double reference = 0,
            TextAnchors anchor = TextAnchors.Left,
            TextBaselines textBaseline = TextBaselines.Top,
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter,
            LineDash?  lineDash = null,
            string tag = null )
```

Stroke a text string along a GraphicsPath.

Parameters

path	The GraphicsPath along which the text will flow.
text	The string to draw.
font	The font with which to draw the text.
strokeColour	The Brush with which to stroke the text.
lineWidth	The width of the line with which the text is stroked.
lineCap	The line cap to use to stroke the text.
lineJoin	The line join to use to stroke the text.
lineDash	The line dash to use to stroke the text.
reference	The (relative) starting point on the path starting from which the text should be drawn (0 is the start of the path, 1 is the end of the path).
anchor	The anchor in the text string that will correspond to the point specified by the *reference* .
textBaseline	The text baseline (determines which the position of the text in relation to the *path* .
tag	A tag to identify the stroked text.

Definition at line 374 of file Graphics.Text.cs.

7.65.2.53 StrokeTextUnderline() [1/4]

```
void VectSharp.Graphics.StrokeTextUnderline (
            double originX,
            double originY,
            IEnumerable< FormattedText > text,
            Brush strokeColour,
            TextBaselines textBaseline = TextBaselines.Top,
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter,
            LineDash?  lineDash = null,
            string tag = null )
```

Stroke the underline of the specified formatted text string.

Parameters

originX	The horizontal coordinate of the text origin.
originY	The vertical coordinate of the text origin. See *textBaseline* .
text	The FormattedText to draw.
strokeColour	The default Brush with which to stroke the underline.
lineWidth	The width of the line with which the underline is stroked.
lineCap	The line cap to use to stroke the underline.
lineJoin	The line join to use to stroke the underline.
lineDash	The line dash to use to stroke the underline.
textBaseline	The text baseline (determines what *originY* represents).
tag	A tag to identify the stroked underline.

Definition at line 1028 of file Graphics.Text.cs.

7.65.2.54 StrokeTextUnderline() [2/4]

```
void VectSharp.Graphics.StrokeTextUnderline (
            double originX,
            double originY,
            string text,
            Font font,
            Brush strokeColour,
            TextBaselines textBaseline = TextBaselines.Top,
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter,
            LineDash?  lineDash = null,
            string tag = null )
```

Stroke the underline of the specified text string.

Parameters

originX	The horizontal coordinate of the text origin.
originY	The vertical coordinate of the text origin. See *textBaseline* .
text	The string whose underline will be drawn.
font	The font with which to draw the text.
strokeColour	The Brush with which to stroke the underline.
lineWidth	The width of the line with which the underline is stroked.
lineCap	The line cap to use to stroke the underline.
lineJoin	The line join to use to stroke the underline.
lineDash	The line dash to use to stroke the underline.
textBaseline	The text baseline (determines what *originY* represents).
tag	A tag to identify the stroked underline.

Definition at line 871 of file Graphics.Text.cs.

7.65.2.55 StrokeTextUnderline() [3/4]

```
void VectSharp.Graphics.StrokeTextUnderline (
            Point origin,
            IEnumerable< FormattedText > text,
            Brush strokeColour,
            TextBaselines textBaseline = TextBaselines.Top,
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter,
            LineDash?  lineDash = null,
            string tag = null )
```

Stroke the underline of the specified formatted text string.

Parameters

origin	The text origin. See *textBaseline* .
text	The FormattedText to draw.
strokeColour	The default Brush with which to stroke the underline.
lineWidth	The width of the line with which the underline is stroked.
lineCap	The line cap to use to stroke the underline.
lineJoin	The line join to use to stroke the underline.
lineDash	The line dash to use to stroke the underline.
textBaseline	The text baseline (determines what the vertical component of *origin* represents).
tag	A tag to identify the stroked underline.

Definition at line 1045 of file Graphics.Text.cs.

7.65.2.56 StrokeTextUnderline() [4/4]

```
void VectSharp.Graphics.StrokeTextUnderline (
            Point origin,
            string text,
            Font font,
            Brush strokeColour,
            TextBaselines textBaseline = TextBaselines.Top,
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter,
            LineDash?  lineDash = null,
            string tag = null )
```

Stroke the underline of the specified text string.

Parameters

origin	The text origin. See *textBaseline* .
text	The string whose underline will be drawn.
font	The font with which to draw the text.
strokeColour	The Brush with which to stroke the underline.
lineWidth	The width of the line with which the underline is stroked.
lineCap	The line cap to use to stroke the underline.
lineJoin	The line join to use to stroke the underline.
lineDash	The line dash to use to stroke the underline.
textBaseline	The text baseline (determines what the vertical component of *origin* represents).
tag	A tag to identify the stroked underline.

Definition at line 889 of file Graphics.Text.cs.

7.65.2.57 Transform() [1/3]

```
void VectSharp.Graphics.Transform (
            double a,
            double b,
            double c,
            double d,
            double e,
            double f,
            string tag = null )
```

Transform the coordinate system with the specified transformation matrix [[a, c, e], [b, d, f], [0, 0, 1]].

Parameters

a	The first element of the first column.
b	The second element of the first column.
c	The first element of the second column.
d	The second element of the second column.
e	The first element of the third column.
f	The second element of the third column.
tag	A tag to identify the transform.

Definition at line 424 of file Graphics.cs.

7.65.2.58 Transform() [2/3]

```
Graphics VectSharp.Graphics.Transform (
            Func< Point, Point > transformationFunction,
            double linearisationResolution )
```

Creates a new Graphics object in which all the graphics actions have been transformed using an arbitrary transformation function. Raster images are replaced by grey rectangles.

Parameters

transformationFunction	An arbitrary transformation function.
linearisationResolution	The resolution that will be used to linearise curve segments.

Returns

A new Graphics object in which all graphics actions have been linearised and transformed using the *transformationFunction* .

Definition at line 1072 of file Graphics.cs.

7.65.2.59 Transform() [3/3]

```
Graphics VectSharp.Graphics.Transform (
            Func< Point, Point > transformationFunction,
            double linearisationResolution,
            double maxSegmentLength )
```

Creates a new Graphics object in which all the graphics actions have been transformed using an arbitrary transformation function. Raster images are replaced by grey rectangles.

Parameters

transformationFunction	An arbitrary transformation function.
linearisationResolution	The resolution that will be used to linearise curve segments.
maxSegmentLength	The maximum length of line segments.

Returns

A new Graphics object in which all graphics actions have been linearised and transformed using the *transformationFunction* .

Definition at line 1271 of file Graphics.cs.

7.65.2.60 Translate() [1/2]

```
void VectSharp.Graphics.Translate (
            double x,
            double y,
            string tag = null )
```

Translate the coordinate system origin.

Parameters

x	The horizontal translation.
y	The vertical translation.
tag	A tag to identify the transform.

Definition at line 436 of file Graphics.cs.

7.65.2.61 Translate() [2/2]

```
void VectSharp.Graphics.Translate (
            Point delta,
            string tag = null )
```

Translate the coordinate system origin.

Parameters

delta	The new origin point.
tag	A tag to identify the transform.

Definition at line 446 of file Graphics.cs.

7.65.2.62 TryRasterise()

```
bool VectSharp.Graphics.TryRasterise (
            Rectangle region,
            double scale,
            bool interpolate,
            out RasterImage output )
```

Tries to rasterise specified region of this Graphics object using the default rasterisation method.

Parameters

region	The region of the Graphics to rasterise.
scale	The scale at which the image is rasterised.
interpolate	Determines whether the resulting RasterImage should be interpolated or not.
output	When this method returns, this will contain the rasterised image (or `null` if the image could not be rasterised.

Returns

`true` if the image could be rasterised; `false` if it could not be rasterised.

Definition at line 1566 of file Graphics.cs.

7.65.3 Member Data Documentation

7.65.3.1 RasterisationMethod

```
Func<Graphics, Rectangle, double, bool, RasterImage> VectSharp.Graphics.RasterisationMethod =
null  [static]
```

A method that is used to rasterise a region of a Graphics object. Set this to `null` if you wish to use the default rasterisation methods (implemented by either VectSharp.Raster, or VectSharp.Raster.ImageSharp). You will have to provide your own implementation of this method if neither VectSharp.Raster nor VectSharp.Raster.ImageSharp are referenced by your project. The first argument of this method is the Graphics to be rasterised, the second is a Rectangle representing the region to rasterise, the third is a double representing the scale, and the third is a boolean value indicating whether the resulting RasterImage should be interpolated.

Definition at line 1556 of file Graphics.cs.

7.65.4 Property Documentation

7.65.4.1 DefaultFillRule

```
FillRule VectSharp.Graphics.DefaultFillRule = FillRule.NonZeroWinding  [get], [set]
```

The default fill rule.

Definition at line 291 of file Graphics.cs.

7.65.4.2 UnbalancedStackAction

```
UnbalancedStackActions VectSharp.Graphics.UnbalancedStackAction = UnbalancedStackActions.Throw
[static], [get], [set]
```

Determines how an unbalanced graphics state stack (which occurs if the number of calls to Save and Restore is not equal) will be treated. The default is UnbalancedStackActions.Throw.

Definition at line 284 of file Graphics.cs.

7.65.4.3 UseUniqueTags

```
bool VectSharp.Graphics.UseUniqueTags = true  [get], [set]
```

Determines whether unique tags should be used for graphics actions that create multiple objects (e.g. drawing text).

Definition at line 319 of file Graphics.cs.

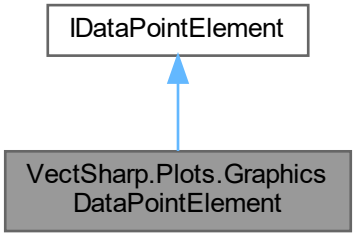
The documentation for this class was generated from the following files:

- VectSharp/Graphics.cs
- VectSharp/Graphics.Text.cs

7.66 VectSharp.Plots.GraphicsDataPointElement Class Reference

A symbol defined by a VectSharp.Graphics object.

Inheritance diagram for VectSharp.Plots.GraphicsDataPointElement:



Public Member Functions

- void Plot (Graphics target, PlotElementPresentationAttributes presentationAttributes, string tag)

  Draw the symbol on the plot.

Parameters

target	*The Graphics object on which to draw. It is assumed that it has been transformed so that the symbol can be drawn centred at (0, 0)*
presentationAttributes	*Presentation attributes determining the appearance of the symbol.*
tag	*A tag to identify the symbol in the plot.*

- GraphicsDataPointElement (Graphics graphics)

  Creates a new GraphicsDataPointElement instance.

Properties

- Graphics Graphics `[get, set]`

  The VectSharp.Graphics object that will be copied on the plot.

7.66.1 Detailed Description

A symbol defined by a VectSharp.Graphics object.

Definition at line 96 of file DataPoints.cs.

7.66.2 Constructor & Destructor Documentation

7.66.2.1 GraphicsDataPointElement()

```
VectSharp.Plots.GraphicsDataPointElement.GraphicsDataPointElement (
            Graphics graphics )
```

Creates a new GraphicsDataPointElement instance.

Parameters

graphics	

Definition at line 113 of file DataPoints.cs.

7.66.3 Member Function Documentation

7.66.3.1 Plot()

```
void VectSharp.Plots.GraphicsDataPointElement.Plot (
            Graphics target,
            PlotElementPresentationAttributes presentationAttributes,
            string tag )
```

Draw the symbol on the plot.

Parameters

target	The Graphics object on which to draw. It is assumed that it has been transformed so that the symbol can be drawn centred at (0, 0)
presentationAttributes	Presentation attributes determining the appearance of the symbol.
tag	A tag to identify the symbol in the plot.

Implements VectSharp.Plots.IDataPointElement.

Definition at line 104 of file DataPoints.cs.

7.66.4 Property Documentation

7.66.4.1 Graphics

```
Graphics VectSharp.Plots.GraphicsDataPointElement.Graphics  [get], [set]
```

The VectSharp.Graphics object that will be copied on the plot.

Definition at line 101 of file DataPoints.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/DataPoints.cs

7.67 VectSharp.GraphicsPath Class Reference

Represents a graphics path that can be filled or stroked.

Public Member Functions

- GraphicsPath MoveTo (Point p)

  Move the current point without tracing a segment from the previous point.
- GraphicsPath MoveTo (double x, double y)

  Move the current point without tracing a segment from the previous point.
- GraphicsPath LineTo (Point p)

  Move the current point and trace a segment from the previous point.
- GraphicsPath LineTo (double x, double y)

  Move the current point and trace a segment from the previous point.
- GraphicsPath Arc (Point center, double radius, double startAngle, double endAngle)

  Trace an arc segment from a circle with the specified center and radius , starting at startAngle and ending at endAngle . The current point is updated to the end point of the arc.
- GraphicsPath Arc (double centerX, double centerY, double radius, double startAngle, double endAngle)

  Trace an arc segment from a circle with the specified center and radius , starting at startAngle and ending at endAngle . The current point is updated to the end point of the arc.
- GraphicsPath EllipticalArc (double radiusX, double radiusY, double axisAngle, bool largeArc, bool sweep↩ Clockwise, Point endPoint)

  Trace an arc from an ellipse with the specified radii, rotated by axisAngle with respect to the x-axis, starting at the current point and ending at the endPoint .
- GraphicsPath CubicBezierTo (Point control1, Point control2, Point endPoint)

  Trace a cubic Bezier curve from the current point to a destination point, with two control points. The current point is updated to the end point of the Bezier curve.
- GraphicsPath CubicBezierTo (double control1X, double control1Y, double control2X, double control2Y, double endPointX, double endPointY)

  Trace a cubic Bezier curve from the current point to a destination point, with two control points. The current point is updated to the end point of the Bezier curve.
- GraphicsPath QuadraticBezierTo (Point control, Point endPoint)

  Trace a quadratic Bezier curve from the current point to a destination point, with a single control point. The current point is updated to the end point of the Bezier curve.
- GraphicsPath QuadraticBezierTo (double controlX, double controlY, double endPointX, double endPointY)

  Trace a quadratic Bezier curve from the current point to a destination point, with a single control point. The current point is updated to the end point of the Bezier curve.
- GraphicsPath Close ()

  Trace a segment from the current point to the start point of the figure and flag the figure as closed.
- GraphicsPath AddText (double originX, double originY, string text, Font font, TextBaselines text↩ Baseline=TextBaselines.Top)

  Add the contour of a text string to the current path.
- GraphicsPath AddText (Point origin, string text, Font font, TextBaselines textBaseline=TextBaselines.Top)

  Add the contour of a text string to the current path.
- GraphicsPath AddTextOnPath (GraphicsPath path, string text, Font font, double reference=0, TextAnchors anchor=TextAnchors.Left, TextBaselines textBaseline=TextBaselines.Top)

  Add the contour of a text string flowing along a GraphicsPath to the current path.
- GraphicsPath AddTextUnderline (Point origin, string text, Font font, TextBaselines textBaseline=Text↩ Baselines.Top)

  Add the contour of the underline of the specified text string to the current path.
- GraphicsPath AddSmoothSpline (params Point[ ] points)

  Adds a smooth spline composed of cubic bezier segments that pass through the specified points.
- GraphicsPath AddPath (GraphicsPath path)

  Adds another GraphicsPath to the current GraphicsPath.
- double MeasureLength ()

  Measures the length of the GraphicsPath.
- Point GetPointAtRelative (double position)

Gets the point at the relative position specified on the GraphicsPath.

- Point GetPointAtAbsolute (double length)

  Gets the point at the absolute position specified on the GraphicsPath.
- Point GetTangentAtRelative (double position)

  Gets the tangent to the point at the relative position specified on the GraphicsPath.
- Point GetTangentAtAbsolute (double length)

  Gets the tangent to the point at the absolute position specified on the GraphicsPath.
- Point GetNormalAtAbsolute (double length)

  Gets the normal to the point at the absolute position specified on the GraphicsPath.
- Point GetNormalAtRelative (double position)

  Gets the normal to the point at the relative position specified on the GraphicsPath.
- GraphicsPath Linearise (double resolution)

  Linearises a GraphicsPath, replacing curve segments with series of line segments that approximate them.
- GraphicsPath Discretise (double resolution)

  Discretises a GraphicsPath, replacing curve segments with series of line segments that approximate them and ensuring that all line segments are shorter than the specified resolution .
- GraphicsPath Flatten (double flatness)

  Flattens a GraphicsPath, replacing curve segments with series of line segments that approximate them, ensuring the specified maximum deviation from the original path.
- IEnumerable< List< Point > > GetPoints ()

  Gets a collection of the end points of all the segments in the GraphicsPath, divided by figure.
- IEnumerable< GraphicsPath > GetFigures ()

  Gets a collection of all the figures in the GraphicsPath, returned as individual GraphicsPaths.
- IEnumerable< List< Point > > GetLinearisationPointsNormals (double resolution)

  Gets a collection of the tangents at the end point of the segments in which the GraphicsPath would be linearised, divided by figure.
- IEnumerable< GraphicsPath > Triangulate (double resolution, bool clockwise)

  Divides a GraphicsPath into triangles.
- GraphicsPath Transform (Func< Point, Point > transformationFunction)

  Transforms all of the Points in the GraphicsPath with an arbitrary transformation function.
- Rectangle GetBounds ()

  Compute the rectangular bounds of the path.
- GraphicsPath Reverse ()

  Reverses the GraphicsPath.
- GraphicsPath GetStroke (double lineWidth=1, LineCaps lineCap=LineCaps.Butt, LineJoins lineJoin=LineJoins.Miter)

  Returns a GraphicsPath representing the stroke of the current GraphicsPath.
- bool ContainsPoint (Point point, FillRule fillRule)

  Determines whether the specified point falls within the current GraphicsPath.

Properties

- List< Segment > Segments = new List() `[get, set]`

  The segments that make up the path.

7.67.1 Detailed Description

Represents a graphics path that can be filled or stroked.

Definition at line 28 of file GraphicsPath.cs.

7.67.2 Member Function Documentation

7.67.2.1 AddPath()

```
GraphicsPath VectSharp.GraphicsPath.AddPath (
            GraphicsPath path )
```

Adds another GraphicsPath to the current GraphicsPath.

Parameters

path	The existing GraphicsPath that should be added to the current GraphicsPath.

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 1210 of file GraphicsPath.cs.

7.67.2.2 AddSmoothSpline()

```
GraphicsPath VectSharp.GraphicsPath.AddSmoothSpline (
            params Point[] points )
```

Adds a smooth spline composed of cubic bezier segments that pass through the specified points.

Parameters

points	The points through which the spline should pass.

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 1178 of file GraphicsPath.cs.

7.67.2.3 AddText() [1/2]

```
GraphicsPath VectSharp.GraphicsPath.AddText (
            double originX,
            double originY,
```

```
            string text,
            Font font,
            TextBaselines textBaseline = TextBaselines.Top )
```

Add the contour of a text string to the current path.

Parameters

originX	The horizontal coordinate of the text origin.
originY	The vertical coordinate of the text origin. See *textBaseline* .
text	The string to draw.
font	The font with which to draw the text.
textBaseline	The text baseline (determines what *originY* represents).

///

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 358 of file GraphicsPath.cs.

7.67.2.4 AddText() [2/2]

```
GraphicsPath VectSharp.GraphicsPath.AddText (
            Point origin,
            string text,
            Font font,
            TextBaselines textBaseline = TextBaselines.Top )
```

Add the contour of a text string to the current path.

Parameters

origin	The text origin. See *textBaseline* .
text	The string to draw.
font	The font with which to draw the text.
textBaseline	The text baseline (determines what the vertical component of *origin* represents).

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 371 of file GraphicsPath.cs.

7.67.2.5 AddTextOnPath()

```
GraphicsPath VectSharp.GraphicsPath.AddTextOnPath (
            GraphicsPath path,
            string text,
            Font font,
```

```
double reference = 0,
TextAnchors anchor = TextAnchors.Left,
TextBaselines textBaseline = TextBaselines.Top )
```

Add the contour of a text string flowing along a GraphicsPath to the current path.

Parameters

path	The GraphicsPath along which the text will flow.
text	The string to draw.
font	The font with which to draw the text.
reference	The (relative) starting point on the path starting from which the text should be drawn (0 is the start of the path, 1 is the end of the path).
anchor	The anchor in the text string that will correspond to the point specified by the *reference* .
textBaseline	The text baseline (determines which the position of the text in relation to the *path* .

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 476 of file GraphicsPath.cs.

7.67.2.6 AddTextUnderline()

```
GraphicsPath VectSharp.GraphicsPath.AddTextUnderline (
            Point origin,
            string text,
            Font font,
            TextBaselines textBaseline = TextBaselines.Top )
```

Add the contour of the underline of the specified text string to the current path.

Parameters

origin	The text origin. See *textBaseline* .
text	The string whose underline will be drawn.
font	The font with which to draw the text.
textBaseline	The text baseline (determines what the vertical component of *origin* represents).

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 624 of file GraphicsPath.cs.

7.67.2.7 Arc() [1/2]

```
GraphicsPath VectSharp.GraphicsPath.Arc (
            double centerX,
            double centerY,
            double radius,
            double startAngle,
            double endAngle )
```

Trace an arc segment from a circle with the specified center and *radius* , starting at *startAngle* and ending at *endAngle* . The current point is updated to the end point of the arc.

Parameters

centerX	The horizontal coordinate of the center of the arc.
centerY	The vertical coordinate of the center of the arc.
radius	The radius of the arc.
startAngle	The start angle (in radians) of the arc.
endAngle	The end angle (in radians) of the arc.

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 126 of file GraphicsPath.cs.

7.67.2.8 Arc() [2/2]

```
GraphicsPath VectSharp.GraphicsPath.Arc (
            Point center,
            double radius,
            double startAngle,
            double endAngle )
```

Trace an arc segment from a circle with the specified *center* and *radius* , starting at *startAngle* and ending at *endAngle* . The current point is updated to the end point of the arc.

Parameters

center	The center of the arc.
radius	The radius of the arc.
startAngle	The start angle (in radians) of the arc.
endAngle	The end angle (in radians) of the arc.

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 103 of file GraphicsPath.cs.

7.67.2.9 Close()

```
GraphicsPath VectSharp.GraphicsPath.Close ( )
```

Trace a segment from the current point to the start point of the figure and flag the figure as closed.

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 341 of file GraphicsPath.cs.

7.67.2.10 ContainsPoint()

```
bool VectSharp.GraphicsPath.ContainsPoint (
            Point point,
            FillRule fillRule )
```

Determines whether the specified *point* falls within the current GraphicsPath.

Parameters

point	The Point being analysed.
fillRule	The rule to use to determine whether a point is inside or outside of a the GraphicsPath.

Returns

`true` if the *point* is inside the GraphicsPath, or `false` if it is outside. If the point lies exactly on the GraphicsPath (or very close to it), the result may be either `true` or `false`, depending on numerical approximations.

Definition at line 3599 of file GraphicsPath.cs.

7.67.2.11 CubicBezierTo() [1/2]

```
GraphicsPath VectSharp.GraphicsPath.CubicBezierTo (
            double control1X,
            double control1Y,
            double control2X,
            double control2Y,
            double endPointX,
            double endPointY )
```

Trace a cubic Bezier curve from the current point to a destination point, with two control points. The current point is updated to the end point of the Bezier curve.

Parameters

control1X	The horizontal coordinate of the first control point.
control1Y	The vertical coordinate of the first control point.
control2X	The horizontal coordinate of the second control point.
control2Y	The vertical coordinate of the second control point.
endPointX	The horizontal coordinate of the destination point.
endPointY	The vertical coordinate of the destination point.

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 285 of file GraphicsPath.cs.

7.67.2.12 CubicBezierTo() [2/2]

```
GraphicsPath VectSharp.GraphicsPath.CubicBezierTo (
            Point control1,
            Point control2,
            Point endPoint )
```

Trace a cubic Bezier curve from the current point to a destination point, with two control points. The current point is updated to the end point of the Bezier curve.

Parameters

control1	The first control point.
control2	The second control point.
endPoint	The destination point.

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 261 of file GraphicsPath.cs.

7.67.2.13 Discretise()

```
GraphicsPath VectSharp.GraphicsPath.Discretise (
            double resolution )
```

Discretises a GraphicsPath, replacing curve segments with series of line segments that approximate them and ensuring that all line segments are shorter than the specified *resolution* .

Parameters

resolution	The maximum length (in absolute units) of line segments in the resulting GraphicsPath.

Returns

A GraphicsPath composed only of linear segments that are shorter than *resolution* and approximate the current GraphicsPath.

Definition at line 1966 of file GraphicsPath.cs.

7.67.2.14 EllipticalArc()

```
GraphicsPath VectSharp.GraphicsPath.EllipticalArc (
            double radiusX,
            double radiusY,
            double axisAngle,
            bool largeArc,
            bool sweepClockwise,
            Point endPoint )
```

Trace an arc from an ellipse with the specified radii, rotated by *axisAngle* with respect to the x-axis, starting at the current point and ending at the *endPoint* .

Parameters

radiusX	The horizontal radius of the ellipse.
radiusY	The vertical radius of the ellipse.
axisAngle	The angle of the horizontal axis of the ellipse with respect to the horizontal axis.
largeArc	Determines whether the large or the small arc is drawn.
sweepClockwise	Determines whether the clockwise or anticlockwise arc is drawn.
endPoint	The end point of the arc.

Returns

Definition at line 142 of file GraphicsPath.cs.

7.67.2.15 Flatten()

```
GraphicsPath VectSharp.GraphicsPath.Flatten (
            double flatness )
```

Flattens a GraphicsPath, replacing curve segments with series of line segments that approximate them, ensuring the specified maximum deviation from the original path.

Parameters

flatness	The maximum deviation from the original path.

Returns

A GraphicsPath composed only of linear segments that approximates the current GraphicsPath.

Definition at line 1997 of file GraphicsPath.cs.

7.67.2.16 GetBounds()

```
Rectangle VectSharp.GraphicsPath.GetBounds ( )
```

Compute the rectangular bounds of the path.

Returns

The smallest Rectangle that contains the path.

Definition at line 3037 of file GraphicsPath.cs.

7.67.2.17 GetFigures()

```
IEnumerable< GraphicsPath > VectSharp.GraphicsPath.GetFigures ( )
```

Gets a collection of all the figures in the GraphicsPath, returned as individual GraphicsPaths.

Returns

A collection of all the figures in the GraphicsPath, returned as individual GraphicsPaths.

Definition at line 2093 of file GraphicsPath.cs.

7.67.2.18 GetLinearisationPointsNormals()

```
IEnumerable< List< Point > > VectSharp.GraphicsPath.GetLinearisationPointsNormals (
            double resolution )
```

Gets a collection of the tangents at the end point of the segments in which the GraphicsPath would be linearised, divided by figure.

Parameters

resolution	The absolute length between successive samples in curve segments.

Returns

A collection of the tangents at the end point of the segments in which the GraphicsPath would be linearised, divided by figure.

Definition at line 2136 of file GraphicsPath.cs.

7.67.2.19 GetNormalAtAbsolute()

```
Point VectSharp.GraphicsPath.GetNormalAtAbsolute (
            double length )
```

Gets the normal to the point at the absolute position specified on the GraphicsPath.

Parameters

length	The distance to the point from the start of the GraphicsPath.

Returns

The normal to the point at the specified position.

Definition at line 1888 of file GraphicsPath.cs.

7.67.2.20 GetNormalAtRelative()

```
Point VectSharp.GraphicsPath.GetNormalAtRelative (
            double position )
```

Gets the normal to the point at the relative position specified on the GraphicsPath.

Parameters

position	The position on the GraphicsPath (0 is the start of the path, 1 is the end of the path).

Returns

The normal to the point at the specified position.

Definition at line 1899 of file GraphicsPath.cs.

7.67.2.21 GetPointAtAbsolute()

```
Point VectSharp.GraphicsPath.GetPointAtAbsolute (
            double length )
```

Gets the point at the absolute position specified on the GraphicsPath.

Parameters

length	The distance to the point from the start of the GraphicsPath.

Returns

The point at the specified position.

Definition at line 1304 of file GraphicsPath.cs.

7.67.2.22 GetPointAtRelative()

```
Point VectSharp.GraphicsPath.GetPointAtRelative (
            double position )
```

Gets the point at the relative position specified on the GraphicsPath.

Parameters

position	The position on the GraphicsPath (0 is the start of the path, 1 is the end of the path).

Returns

The point at the specified position.

Definition at line 1294 of file GraphicsPath.cs.

7.67.2.23 GetPoints()

```
IEnumerable< List< Point > > VectSharp.GraphicsPath.GetPoints ( )
```

Gets a collection of the end points of all the segments in the GraphicsPath, divided by figure.

Returns

A collection of the end points of all the segments in the GraphicsPath, divided by figure.

Definition at line 2050 of file GraphicsPath.cs.

7.67.2.24 GetStroke()

```
GraphicsPath VectSharp.GraphicsPath.GetStroke (
            double lineWidth = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineJoins lineJoin = LineJoins.Miter )
```

Returns a GraphicsPath representing the stroke of the current GraphicsPath.

Parameters

lineWidth	The thickness of the stroke.
lineCap	The line cap used in the stroke.
lineJoin	The line join used in the stroke.

Returns

A GraphicsPath representing the stroke of the current GraphicsPath.

Definition at line 3445 of file GraphicsPath.cs.

7.67.2.25 GetTangentAtAbsolute()

```
Point VectSharp.GraphicsPath.GetTangentAtAbsolute (
            double length )
```

Gets the tangent to the point at the absolute position specified on the GraphicsPath.

Parameters

length	The distance to the point from the start of the GraphicsPath.

Returns

The tangent to the point at the specified position.

Definition at line 1601 of file GraphicsPath.cs.

7.67.2.26 GetTangentAtRelative()

```
Point VectSharp.GraphicsPath.GetTangentAtRelative (
            double position )
```

Gets the tangent to the point at the relative position specified on the GraphicsPath.

Parameters

position	The position on the GraphicsPath (0 is the start of the path, 1 is the end of the path).

Returns

The tangent to the point at the specified position.

Definition at line 1591 of file GraphicsPath.cs.

7.67.2.27 Linearise()

```
GraphicsPath VectSharp.GraphicsPath.Linearise (
            double resolution )
```

Linearises a GraphicsPath, replacing curve segments with series of line segments that approximate them.

Parameters

resolution	The absolute length between successive samples in curve segments.

Returns

A GraphicsPath composed only of linear segments that approximates the current GraphicsPath.

Definition at line 1911 of file GraphicsPath.cs.

7.67.2.28 LineTo() [1/2]

```
GraphicsPath VectSharp.GraphicsPath.LineTo (
            double x,
            double y )
```

Move the current point and trace a segment from the previous point.

Parameters

x	The horizontal coordinate of the new point.
y	The vertical coordinate of the new point.

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 88 of file GraphicsPath.cs.

7.67.2.29 LineTo() [2/2]

```
GraphicsPath VectSharp.GraphicsPath.LineTo (
            Point p )
```

Move the current point and trace a segment from the previous point.

Parameters

p	The new point.

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 66 of file GraphicsPath.cs.

7.67.2.30 MeasureLength()

```
double VectSharp.GraphicsPath.MeasureLength ( )
```

Measures the length of the GraphicsPath.

Returns

The length of the GraphicsPath

Definition at line 1222 of file GraphicsPath.cs.

7.67.2.31 MoveTo() [1/2]

```
GraphicsPath VectSharp.GraphicsPath.MoveTo (
            double x,
            double y )
```

Move the current point without tracing a segment from the previous point.

Parameters

x	The horizontal coordinate of the new point.
y	The vertical coordinate of the new point.

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 55 of file GraphicsPath.cs.

7.67.2.32 MoveTo() [2/2]

```
GraphicsPath VectSharp.GraphicsPath.MoveTo (
            Point p )
```

Move the current point without tracing a segment from the previous point.

Parameters

p	The new point.

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 41 of file GraphicsPath.cs.

7.67.2.33 QuadraticBezierTo() [1/2]

```
GraphicsPath VectSharp.GraphicsPath.QuadraticBezierTo (
            double controlX,
            double controlY,
            double endPointX,
            double endPointY )
```

Trace a quadratic Bezier curve from the current point to a destination point, with a single control point. The current point is updated to the end point of the Bezier curve.

Parameters

controlX	The horizontal coordinate of the control point.
controlY	The vertical coordinate of the control point.
endPointX	The horizontal coordinate of the destination point.
endPointY	The vertical coordinate of the destination point.

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 331 of file GraphicsPath.cs.

7.67.2.34 QuadraticBezierTo() [2/2]

```
GraphicsPath VectSharp.GraphicsPath.QuadraticBezierTo (
            Point control,
            Point endPoint )
```

Trace a quadratic Bezier curve from the current point to a destination point, with a single control point. The current point is updated to the end point of the Bezier curve.

Parameters

control	The control point.
endPoint	The destination point.

Returns

The GraphicsPath, to allow for chained calls.

Definition at line 298 of file GraphicsPath.cs.

7.67.2.35 Reverse()

```
GraphicsPath VectSharp.GraphicsPath.Reverse ( )
```

Reverses the GraphicsPath.

Returns

The reversed GraphicsPath.

Definition at line 3335 of file GraphicsPath.cs.

7.67.2.36 Transform()

```
GraphicsPath VectSharp.GraphicsPath.Transform (
            Func< Point, Point > transformationFunction )
```

Transforms all of the Points in the GraphicsPath with an arbitrary transformation function.

Parameters

transformationFunction	An arbitrary transformation function.

Returns

A new GraphicsPath in which all points have been replaced using the *transformationFunction* .

Definition at line 3018 of file GraphicsPath.cs.

7.67.2.37 Triangulate()

```
IEnumerable< GraphicsPath > VectSharp.GraphicsPath.Triangulate (
            double resolution,
            bool clockwise )
```

Divides a GraphicsPath into triangles.

Parameters

resolution	The resolution that will be used to linearise curve segments in the GraphicsPath.
clockwise	If this is `true`, the triangles will have their vertices in a clockwise order, otherwise they will be in anticlockwise order.

Returns

A collection of distinct GraphicsPaths, each representing one triangle.

Definition at line 2230 of file GraphicsPath.cs.

7.67.3 Property Documentation

7.67.3.1 Segments

```
List<Segment> VectSharp.GraphicsPath.Segments = new List<Segment>()  [get], [set]
```

The segments that make up the path.

Definition at line 33 of file GraphicsPath.cs.

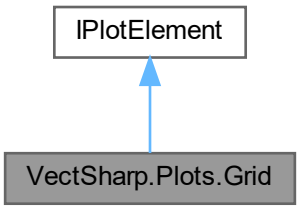
The documentation for this class was generated from the following file:

- VectSharp/GraphicsPath.cs

7.68 VectSharp.Plots.Grid Class Reference

A plot element that draws a grid.

Inheritance diagram for VectSharp.Plots.Grid:



Public Member Functions

- Grid (IReadOnlyList< double > side1Start, IReadOnlyList< double > side1End, IReadOnlyList< double > side2Start, IReadOnlyList< double > side2End, IContinuousCoordinateSystem coordinateSystem)

  Create a new Grid instance.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IReadOnlyList< double > Side1Start `[get, set]`

  The starting point for the first side of the grid.
- IReadOnlyList< double > Side1End `[get, set]`

  The ending point for the first side of the grid.
- IReadOnlyList< double > Side2Start `[get, set]`

  The starting point for the second side of the grid.
- IReadOnlyList< double > Side2End `[get, set]`

  The ending point for the second side of the grid.
- int IntervalCount = 10 `[get, set]`

  The number of intervals between grid lines. Note that the number of grid lines will be one greater than this.
- IContinuousCoordinateSystem CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(220, 220, 220)) } `[get, set]`

  Presentation attributes determining the appearance of the grid.
- string Tag `[get, set]`

  A tag to identify the grid in the plot.

7.68.1 Detailed Description

A plot element that draws a grid.

Definition at line 841 of file Axes.cs.

7.68.2 Constructor & Destructor Documentation

7.68.2.1 Grid()

```
VectSharp.Plots.Grid.Grid (
            IReadOnlyList< double > side1Start,
            IReadOnlyList< double > side1End,
            IReadOnlyList< double > side2Start,
            IReadOnlyList< double > side2End,
            IContinuousCoordinateSystem coordinateSystem )
```

Create a new Grid instance.

Parameters

side1Start	The starting point for the first side of the grid.
side1End	The ending point for the first side of the grid.
side2Start	The starting point for the second side of the grid.
side2End	The ending point for the second side of the grid.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 892 of file Axes.cs.

7.68.3 Member Function Documentation

7.68.3.1 Plot()

```
void VectSharp.Plots.Grid.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 902 of file Axes.cs.

7.68.4 Property Documentation

7.68.4.1 CoordinateSystem

```
IContinuousCoordinateSystem VectSharp.Plots.Grid.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 871 of file Axes.cs.

7.68.4.2 IntervalCount

```
int VectSharp.Plots.Grid.IntervalCount = 10  [get], [set]
```

The number of intervals between grid lines. Note that the number of grid lines will be one greater than this.

Definition at line 866 of file Axes.cs.

7.68.4.3 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.Grid.PresentationAttributes = new PlotElementPresentationAtt
{ Stroke = new SolidColourBrush(Colour.FromRgb(220, 220, 220)) }  [get], [set]
```

Presentation attributes determining the appearance of the grid.

Definition at line 877 of file Axes.cs.

7.68.4.4 Side1End

```
IReadOnlyList<double> VectSharp.Plots.Grid.Side1End  [get], [set]
```

The ending point for the first side of the grid.

Definition at line 851 of file Axes.cs.

7.68.4.5 Side1Start

```
IReadOnlyList<double> VectSharp.Plots.Grid.Side1Start  [get], [set]
```

The starting point for the first side of the grid.

Definition at line 846 of file Axes.cs.

7.68.4.6 Side2End

```
IReadOnlyList<double> VectSharp.Plots.Grid.Side2End  [get], [set]
```

The ending point for the second side of the grid.

Definition at line 861 of file Axes.cs.

7.68.4.7 Side2Start

```
IReadOnlyList<double> VectSharp.Plots.Grid.Side2Start  [get], [set]
```

The starting point for the second side of the grid.

Definition at line 856 of file Axes.cs.

7.68.4.8 Tag

```
string VectSharp.Plots.Grid.Tag  [get], [set]
```

A tag to identify the grid in the plot.

Definition at line 882 of file Axes.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/Axes.cs

7.69 VectSharp.Markdown.HTTPUtils Class Reference

Contains utilities to resolve absolute and relative URIs.

Static Public Member Functions

- static string bool wasDownloaded ResolveEmojiUsingOpenMoji (string emojiUri, MarkdownRenderer renderer)
- static string bool wasDownloaded ResolveImageURI (string uri, string baseUriString)

Static Public Attributes

- static string path

  Uses OpenMoji (https://openmoji.org) to resolve emoji URIs of the form "emoji://{name}_heading:{level}" or "unicode://{unicode}_heading:{level}".

Properties

- static bool LogDownloads = true [get, set]

  Determines whether every file that is downloaded should be logged to the standard error stream.

7.69.1 Detailed Description

Contains utilities to resolve absolute and relative URIs.

Definition at line 246 of file HtmlTag.cs.

7.69.2 Member Function Documentation

7.69.2.1 ResolveEmojiUsingOpenMoji()

```
static string bool wasDownloaded VectSharp.Markdown.HTTPUtils.ResolveEmojiUsingOpenMoji (
            string emojiUri,
            MarkdownRenderer renderer ) [static]
```

Definition at line 259 of file HtmlTag.cs.

7.69.2.2 ResolveImageURI()

```
static string bool wasDownloaded VectSharp.Markdown.HTTPUtils.ResolveImageURI (
            string uri,
            string baseUriString ) [static]
```

Definition at line 335 of file HtmlTag.cs.

7.69.3 Member Data Documentation

7.69.3.1 path

```
static string VectSharp.Markdown.HTTPUtils.path [static]
```

Uses OpenMoji (https://openmoji.org) to resolve emoji URIs of the form "emoji://{name}_heading:{level}" or "unicode://{unicode}_heading:{level}".

Resolves an image Uri, by downloading the image file if necessary. It also takes care of ensuring that the file extension matches the format of the file.

Parameters

emojiUri	The emoji URI.
renderer	The MarkdownRenderer on which the emoji will be drawn.

Returns

A tuple containing the local path of the rendered emoji and a boolean value indicating whether the image should be deleted after the program is done with it.

Parameters

uri	The address of the image.
baseUriString	The base uri to use for relative uris.

Returns

A tuple containing the local path of the image file (either the original image, or a local copy of a remote file) and a boolean value indicating whether the image was fetched from a remote location and should be deleted after the program is done with it.

Definition at line 259 of file HtmlTag.cs.

7.69.4 Property Documentation

7.69.4.1 LogDownloads

```
bool VectSharp.Markdown.HTTPUtils.LogDownloads = true  [static], [get], [set]
```

Determines whether every file that is downloaded should be logged to the standard error stream.

Definition at line 251 of file HtmlTag.cs.

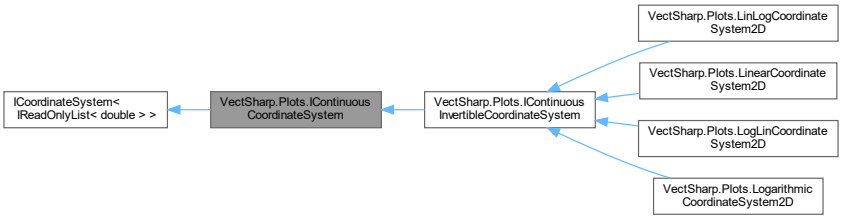
The documentation for this class was generated from the following file:

- VectSharp.Markdown/HtmlTag.cs

7.70 VectSharp.Plots.IContinuousCoordinateSystem Interface Reference

Represents a coordinate system performing continuous transformations.

Inheritance diagram for VectSharp.Plots.IContinuousCoordinateSystem:



Public Member Functions

- bool IsDirectionStraight (IReadOnlyList< double > direction)

  Determines whether points aligned along direction in data space are also aligned along some line in plot space.
- double[ ] GetAround (IReadOnlyList< double > point, IReadOnlyList< double > direction)

  Gets a data element that is arbitrarily close to the specified point , along the specified direction .

Properties

- bool IsLinear `[get]`

  Gets whether the current coordinate system is linear along all directions.
- double[ ] Resolution `[get]`

  The maximum difference between two points in data space that appear arbitrarily close in plot space, or some approximation.

7.70.1 Detailed Description

Represents a coordinate system performing continuous transformations.

Definition at line 90 of file CoordinateSystems.cs.

7.70.2 Member Function Documentation

7.70.2.1 GetAround()

```
double[] VectSharp.Plots.IContinuousCoordinateSystem.GetAround (
            IReadOnlyList< double > point,
            IReadOnlyList< double > direction )
```

Gets a data element that is arbitrarily close to the specified *point* , along the specified *direction* .

Parameters

point	The point close to which the returned data element should be.
direction	The direction (in data space) along which the returned point should be.

Returns

A data element that is arbitrarily close to the specified *point* , along the specified *direction* .

Implemented in VectSharp.Plots.LinearCoordinateSystem2D, VectSharp.Plots.LogarithmicCoordinateSystem2D, VectSharp.Plots.LogLinCoordinateSystem2D, and VectSharp.Plots.LinLogCoordinateSystem2D.

7.70.2.2 IsDirectionStraight()

```
bool VectSharp.Plots.IContinuousCoordinateSystem.IsDirectionStraight (
            IReadOnlyList< double > direction )
```

Determines whether points aligned along *direction* in data space are also aligned along some line in plot space.

Parameters

direction	The direction in data space along which the points should be aligned.

Returns

`true` if any two points aligned along *direction* in data space are also aligned along some line in plot space, `false` otherwise.

Implemented in VectSharp.Plots.LinearCoordinateSystem2D, VectSharp.Plots.LogarithmicCoordinateSystem2D, VectSharp.Plots.LogLinCoordinateSystem2D, and VectSharp.Plots.LinLogCoordinateSystem2D.

7.70.3 Property Documentation

7.70.3.1 IsLinear

```
bool VectSharp.Plots.IContinuousCoordinateSystem.IsLinear  [get]
```

Gets whether the current coordinate system is linear along all directions.

Implemented in VectSharp.Plots.LinearCoordinateSystem2D, VectSharp.Plots.LogarithmicCoordinateSystem2D, VectSharp.Plots.LogLinCoordinateSystem2D, and VectSharp.Plots.LinLogCoordinateSystem2D.

Definition at line 95 of file CoordinateSystems.cs.

7.70.3.2 Resolution

```
double [] VectSharp.Plots.IContinuousCoordinateSystem.Resolution  [get]
```

The maximum difference between two points in data space that appear arbitrarily close in plot space, or some approximation.

Implemented in VectSharp.Plots.LinearCoordinateSystem2D, VectSharp.Plots.LogarithmicCoordinateSystem2D, VectSharp.Plots.LogLinCoordinateSystem2D, and VectSharp.Plots.LinLogCoordinateSystem2D.

Definition at line 107 of file CoordinateSystems.cs.

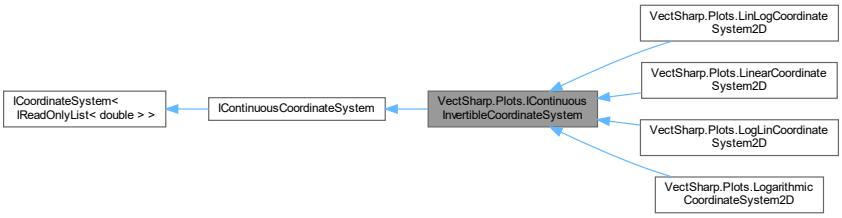
The documentation for this interface was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.71 VectSharp.Plots.IContinuousInvertibleCoordinateSystem Interface Reference

Represents a coordinate system performing continuous invertible transformations.

Inheritance diagram for VectSharp.Plots.IContinuousInvertibleCoordinateSystem:



Public Member Functions

- double[ ] ToDataCoordinates (Point plotPoint)

  Transform a point in plot space back into data space.

Additional Inherited Members

7.71.1 Detailed Description

Represents a coordinate system performing continuous invertible transformations.

Definition at line 121 of file CoordinateSystems.cs.

7.71.2 Member Function Documentation

7.71.2.1 ToDataCoordinates()

```
double[] VectSharp.Plots.IContinuousInvertibleCoordinateSystem.ToDataCoordinates (
            Point plotPoint )
```

Transform a point in plot space back into data space.

Parameters

plotPoint	The point in plot space.

Returns

The point in data space corresponding to the specified point in plot space.

Implemented in VectSharp.Plots.LinearCoordinateSystem2D, VectSharp.Plots.LogarithmicCoordinateSystem2D, VectSharp.Plots.LogLinCoordinateSystem2D, and VectSharp.Plots.LinLogCoordinateSystem2D.

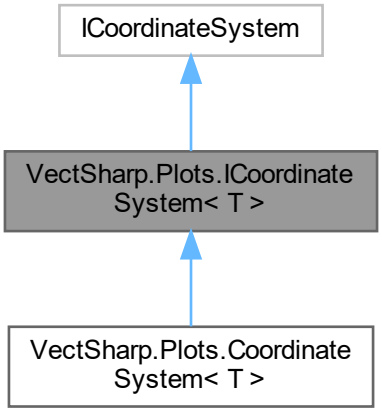
The documentation for this interface was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.72 VectSharp.Plots.ICoordinateSystem< T > Interface Template Reference

Represents a coordinate system.

Inheritance diagram for VectSharp.Plots.ICoordinateSystem< T >:



Public Member Functions

- Point ToPlotCoordinates (T dataPoint)

  Transform the specified dataPoint into a plot Point.

7.72.1 Detailed Description

Represents a coordinate system.

Represents a coordinate system transforming data points of type *T* into plot Points.

Template Parameters

T	The type of data elements.

Definition at line 36 of file CoordinateSystems.cs.

7.72.2 Member Function Documentation

7.72.2.1 ToPlotCoordinates()

```
Point VectSharp.Plots.ICoordinateSystem< T >.ToPlotCoordinates (
            T dataPoint )
```

Transform the specified *dataPoint* into a plot Point.

Parameters

dataPoint	The data point whose coordinates should be determined.

Returns

A Point representing the *dataPoint* in plot space.

Implemented in VectSharp.Plots.CoordinateSystem< T >.

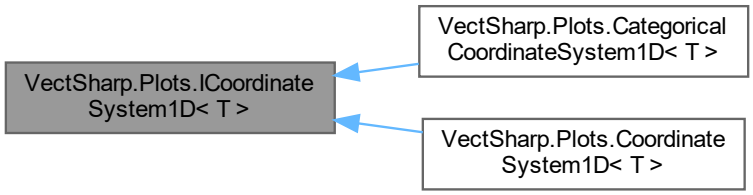
The documentation for this interface was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.73 VectSharp.Plots.ICoordinateSystem1D< T > Interface Template Reference

Represents a coordinate system tranforming data points of type *T* into `double`s.

Inheritance diagram for VectSharp.Plots.ICoordinateSystem1D< T >:



Public Member Functions

- double ToPlotCoordinates (T dataPoint)

  Transform the specified dataPoint into a `double`.

7.73.1 Detailed Description

Represents a coordinate system tranforming data points of type *T* into `double`s.

Template Parameters

T	The type of data elements.

Definition at line 50 of file CoordinateSystems.cs.

7.73.2 Member Function Documentation

7.73.2.1 ToPlotCoordinates()

```
double VectSharp.Plots.ICoordinateSystem1D< T >.ToPlotCoordinates (
            T dataPoint )
```

Transform the specified *dataPoint* into a `double`.

Parameters

dataPoint	The data point whose coordinates should be determined.

Returns

A `double` representing the *dataPoint* in plot space.

Implemented in VectSharp.Plots.CoordinateSystem1D< T >, and VectSharp.Plots.CategoricalCoordinateSystem1D< T >.

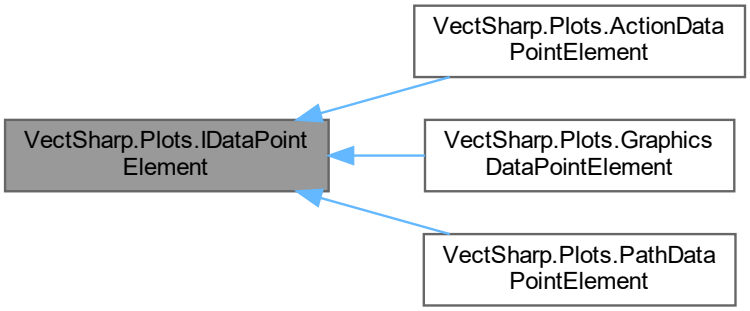
The documentation for this interface was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.74 VectSharp.Plots.IDataPointElement Interface Reference

Represents a symbol that can be added to the plot at a specified position.

Inheritance diagram for VectSharp.Plots.IDataPointElement:



Public Member Functions

- void Plot (Graphics target, PlotElementPresentationAttributes presentationAttributes, string tag)

  Draw the symbol on the plot.

7.74.1 Detailed Description

Represents a symbol that can be added to the plot at a specified position.

Definition at line 26 of file DataPoints.cs.

7.74.2 Member Function Documentation

7.74.2.1 Plot()

```
void VectSharp.Plots.IDataPointElement.Plot (
            Graphics target,
            PlotElementPresentationAttributes presentationAttributes,
            string tag )
```

Draw the symbol on the plot.

Parameters

target	The Graphics object on which to draw. It is assumed that it has been transformed so that the symbol can be drawn centred at (0, 0)
presentationAttributes	Presentation attributes determining the appearance of the symbol.
tag	A tag to identify the symbol in the plot.

Implemented in VectSharp.Plots.PathDataPointElement, VectSharp.Plots.GraphicsDataPointElement, and VectSharp.Plots.ActionDataPointElement.

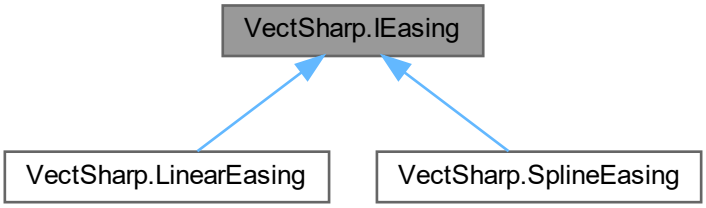
The documentation for this interface was generated from the following file:

- VectSharp.Plots/DataPoints.cs

7.75 VectSharp.IEasing Interface Reference

Describes a function used to transform the transition speed.

Inheritance diagram for VectSharp.IEasing:



Public Member Functions

- double Ease (double value)

  Applies the easing to the specified transition offset.

7.75.1 Detailed Description

Describes a function used to transform the transition speed.

Definition at line 1247 of file Animation.cs.

7.75.2 Member Function Documentation

7.75.2.1 Ease()

```
double VectSharp.IEasing.Ease (
            double value )
```

Applies the easing to the specified transition offset.

Parameters

value	The transition offset (ranging from 0 to 1).

Returns

The eased transition offset value.

Implemented in VectSharp.SplineEasing, and VectSharp.LinearEasing.

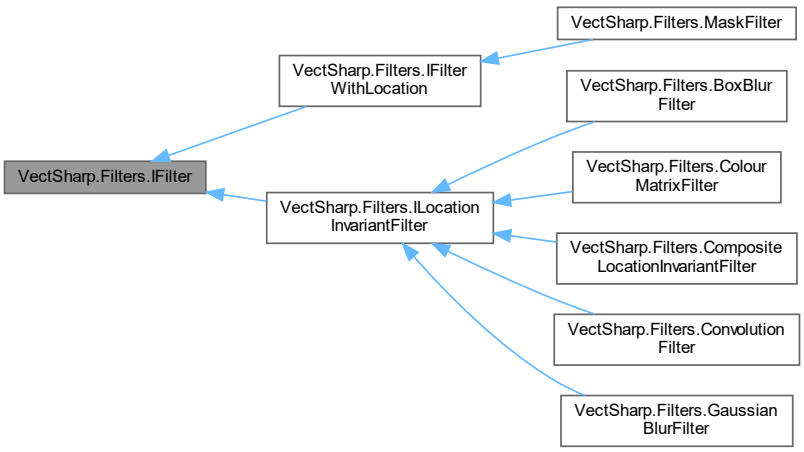
The documentation for this interface was generated from the following file:

- VectSharp/Animation.cs

7.76 VectSharp.Filters.IFilter Interface Reference

Represents a filter. Do not implement this interface directly; instead, implement ILocationInvariantFilter or IFilterWithLocation.

Inheritance diagram for VectSharp.Filters.IFilter:



Properties

- Point TopLeftMargin `[get]`

  Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.

- Point BottomRightMargin `[get]`

  Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.

7.76.1 Detailed Description

Represents a filter. Do not implement this interface directly; instead, implement ILocationInvariantFilter or IFilterWithLocation.

Definition at line 26 of file Filters.cs.

7.76.2 Property Documentation

7.76.2.1 BottomRightMargin

```
Point VectSharp.Filters.IFilter.BottomRightMargin  [get]
```

Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.

Implemented in VectSharp.Filters.BoxBlurFilter, VectSharp.Filters.ColourMatrixFilter, VectSharp.Filters.CompositeLocationInvariantFil VectSharp.Filters.ConvolutionFilter, VectSharp.Filters.GaussianBlurFilter, and VectSharp.Filters.MaskFilter.

Definition at line 36 of file Filters.cs.

7.76.2.2 TopLeftMargin

```
Point VectSharp.Filters.IFilter.TopLeftMargin  [get]
```

Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.

Implemented in VectSharp.Filters.BoxBlurFilter, VectSharp.Filters.ColourMatrixFilter, VectSharp.Filters.CompositeLocationInvariantFil VectSharp.Filters.ConvolutionFilter, VectSharp.Filters.GaussianBlurFilter, and VectSharp.Filters.MaskFilter.

Definition at line 31 of file Filters.cs.

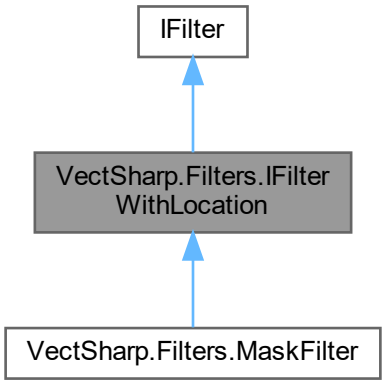
The documentation for this interface was generated from the following file:

- VectSharp/Filters/Filters.cs

7.77 VectSharp.Filters.IFilterWithLocation Interface Reference

Represents a filter whose results depend on the position of the subject image on the graphics surface.

Inheritance diagram for VectSharp.Filters.IFilterWithLocation:



Public Member Functions

- RasterImage Filter (RasterImage image, Rectangle bounds, double scale)
  Applies the filter to a RasterImage.

Additional Inherited Members

7.77.1 Detailed Description

Represents a filter whose results depend on the position of the subject image on the graphics surface.

Definition at line 57 of file Filters.cs.

7.77.2 Member Function Documentation

7.77.2.1 Filter()

```
RasterImage VectSharp.Filters.IFilterWithLocation.Filter (
            RasterImage image,
            Rectangle bounds,
            double scale )
```

Applies the filter to a RasterImage.

Parameters

image	The RasterImage to which the filter will be applied.
bounds	The region on the graphics surface where the image will be drawn.
scale	The scale of the image with respect to the filter.

Returns

A new RasterImage containing the filtered image. The source *image* is left unaltered.

Implemented in VectSharp.Filters.MaskFilter.

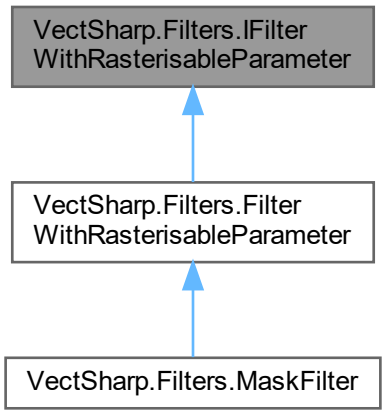
The documentation for this interface was generated from the following file:

- VectSharp/Filters/Filters.cs

7.78 VectSharp.Filters.IFilterWithRasterisableParameter Interface Reference

Represents a filter with a parameter that needs to be rasterised at the same resolution as the subject image prior to applying the filter. The FilterWithRasterisableParameter abstract class provides a default implementation of this interface.

Inheritance diagram for VectSharp.Filters.IFilterWithRasterisableParameter:



Public Member Functions

- void RasteriseParameter (Func< Graphics, Rectangle, double, bool, RasterImage > rasterisationMethod, double scale)

  Rasterises the filter's parameter at the specified scale, using the specified rasterisation method.

7.78.1 Detailed Description

Represents a filter with a parameter that needs to be rasterised at the same resolution as the subject image prior to applying the filter. The FilterWithRasterisableParameter abstract class provides a default implementation of this interface.

Definition at line 74 of file Filters.cs.

7.78.2 Member Function Documentation

7.78.2.1 RasteriseParameter()

```
void VectSharp.Filters.IFilterWithRasterisableParameter.RasteriseParameter (
            Func< Graphics, Rectangle, double, bool, RasterImage > rasterisationMethod,
            double scale )
```

Rasterises the filter's parameter at the specified scale, using the specified rasterisation method.

Parameters

rasterisationMethod	The method used to rasterise the image. The first argument of this method is the Graphics to be rasterised, the second is a Rectangle representing the region to rasterise, the third is a double representing the scale, and the third is a boolean value indicating whether the resulting RasterImage should be interpolated.
scale	The scale factor at which the parameter is rasterised.

Implemented in VectSharp.Filters.FilterWithRasterisableParameter.

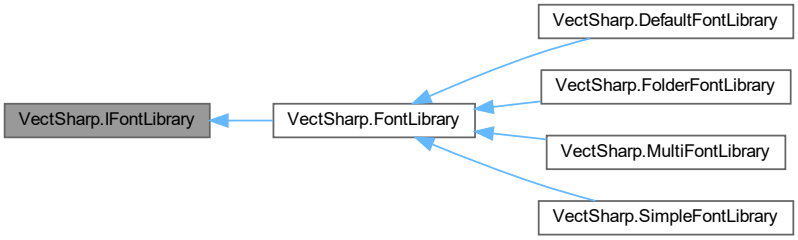
The documentation for this interface was generated from the following file:

- VectSharp/Filters/Filters.cs

7.79 VectSharp.IFontLibrary Interface Reference

Represents a font library with methods to create FontFamily objects from a string or from FontFamily.StandardFontFamilies.

Inheritance diagram for VectSharp.IFontLibrary:



Public Member Functions

- FontFamily ResolveFontFamily (string fontFamily)

  Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised.
- FontFamily ResolveFontFamily (string fontFamily, params string[ ] fallback)

  Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, try to instantiate the font family using the fallback . If none of the fallback family names or true type files are valid, an exception might be raised.
- FontFamily ResolveFontFamily (FontFamily.StandardFontFamilies standardFontFamily)

  Create a new font family from the specified standard font family name.
- FontFamily ResolveFontFamily (string fontFamily, FontFamily.StandardFontFamilies finalFallback, params string[ ] fallback)

  Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, try to instantiate the font family using the fallback . If none of the fallback family names or true type files are valid, instantiate a standard font family using the finalFallback .

7.79.1 Detailed Description

Represents a font library with methods to create FontFamily objects from a string or from FontFamily.StandardFontFamilies.

Definition at line 28 of file FontLibrary.cs.

7.79.2 Member Function Documentation

7.79.2.1 ResolveFontFamily() [1/4]

```
FontFamily VectSharp.IFontLibrary.ResolveFontFamily (
            FontFamily.StandardFontFamilies standardFontFamily )
```

Create a new font family from the specified standard font family name.

Parameters

standardFontFamily	The standard name of the font family.

Returns

A FontFamily object corresponding to the specified font family.

Implemented in VectSharp.FontLibrary, VectSharp.SimpleFontLibrary, VectSharp.DefaultFontLibrary, VectSharp.FolderFontLibrary, and VectSharp.MultiFontLibrary.

7.79.2.2 ResolveFontFamily() [2/4]

```
FontFamily VectSharp.IFontLibrary.ResolveFontFamily (
            string fontFamily )
```

Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised.

Parameters

fontFamily	The name of the font family to create, or the path to a TTF file.

Returns

If the font family name or the true type file is valid, a FontFamily object corresponding to the specified font family.

Implemented in VectSharp.FontLibrary, VectSharp.SimpleFontLibrary, VectSharp.DefaultFontLibrary, VectSharp.FolderFontLibrary, and VectSharp.MultiFontLibrary.

7.79.2.3 ResolveFontFamily() [3/4]

```
FontFamily VectSharp.IFontLibrary.ResolveFontFamily (
            string fontFamily,
            FontFamily.StandardFontFamilies finalFallback,
            params string[] fallback )
```

Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, try to instantiate the font family using the *fallback* . If none of the fallback family names or true type files are valid, instantiate a standard font family using the *finalFallback* .

Parameters

fontFamily	The name of the font family to create, or the path to a TTF file.
fallback	Names of additional font families or TTF files, which will be tried if the first *fontFamily* is not valid.
finalFallback	The standard name of the font family that will be used if none of the fallback families are valid.

Returns

A FontFamily object corresponding to the first of the specified font families that is valid.

Implemented in VectSharp.FontLibrary.

7.79.2.4 ResolveFontFamily() [4/4]

```
FontFamily VectSharp.IFontLibrary.ResolveFontFamily (
            string fontFamily,
            params string[] fallback )
```

Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, try to instantiate the font family using the *fallback* . If none of the fallback family names or true type files are valid, an exception might be raised.

Parameters

fontFamily	The name of the font family to create, or the path to a TTF file.
fallback	Names of additional font families or TTF files, which will be tried if the first *fontFamily* is not valid.

Returns

A FontFamily object corresponding to the first of the specified font families that is valid.

Implemented in VectSharp.FontLibrary.

The documentation for this interface was generated from the following file:

- VectSharp/FontLibrary.cs

7.80 VectSharp.IGraphicsContext Interface Reference

This interface should be implemented by classes intended to provide graphics output capability to a Graphics object.

Public Member Functions

- void **Save** ()

  Save the current transform state (rotation, translation, scale). This should be implemented as a LIFO stack.
- void **Restore** ()

  Restore the previous transform state (rotation, translation, scale). This should be implemented as a LIFO stack.
- void Translate (double x, double y)

  Translate the coordinate system origin.
- void Rotate (double angle)

  Rotate the coordinate system around the origin.
- void Scale (double scaleX, double scaleY)

  Scale the coordinate system with respect to the origin.
- void Transform (double a, double b, double c, double d, double e, double f)

  Transform the coordinate system with the specified transformation matrix [[a, c, e], [b, d, f], [0, 0, 1]].
- void FillText (string text, double x, double y)

  Fill a text string using the current Font and TextBaseline.
- void StrokeText (string text, double x, double y)

  Stroke the outline of a text string using the current Font and TextBaseline.
- void MoveTo (double x, double y)

  Change the current point without drawing a line from the previous point. If necessary, start a new figure.
- void LineTo (double x, double y)

  Draw a line from the previous point to the specified point.
- void **Close** ()

  Close the current figure.
- void **Stroke** ()

Stroke the current path using the current StrokeStyle, LineWidth, LineCap, LineJoin and LineDash.

- void **SetClippingPath** ()

  Set the current clipping path as the intersection of the previous clipping path and the current path.
- void SetFillStyle ((int r, int g, int b, double a) style)

  Set the current FillStyle.
- void SetFillStyle (Brush style)

  Set the current FillStyle.
- void SetStrokeStyle ((int r, int g, int b, double a) style)

  Set the current StrokeStyle.
- void SetStrokeStyle (Brush style)

  Set the current StrokeStyle.
- void CubicBezierTo (double p1X, double p1Y, double p2X, double p2Y, double p3X, double p3Y)

  Add to the current figure a cubic Bezier from the current point to a destination point, with two control points.
- void Rectangle (double x0, double y0, double width, double height)

  Add a rectangle figure to the current path.
- void Fill (FillRule fillRule)

  Fill the current path using the current FillStyle.
- void SetLineDash (LineDash dash)

  Set the current line dash pattern.
- void DrawRasterImage (int sourceX, int sourceY, int sourceWidth, int sourceHeight, double destinationX, double destinationY, double destinationWidth, double destinationHeight, RasterImage image)

  Draw a raster image.
- void DrawFilteredGraphics (Graphics graphics, IFilter filter)

  Draws a Graphics object, applying the specified filter .

Properties

- double Width `[get]`

  Width of the graphic surface.
- double Height `[get]`

  Height of the graphic surface.
- Font Font `[get, set]`

  The current font.
- TextBaselines TextBaseline `[get, set]`

  The current text baseline.
- Brush FillStyle `[get]`

  Current brush used to fill paths.
- Brush StrokeStyle `[get]`

  Current brush used to stroke paths.
- double LineWidth `[get, set]`

  Current line width used to stroke paths.
- LineCaps LineCap `[set]`

  Current line cap used to stroke paths.
- LineJoins LineJoin `[set]`

  Current line join used to stroke paths.
- string Tag `[get, set]`

  The current tag. How this can be used depends on each implementation.

7.80.1 Detailed Description

This interface should be implemented by classes intended to provide graphics output capability to a Graphics object.

Definition at line 36 of file Graphics.cs.

7.80.2 Member Function Documentation

7.80.2.1 CubicBezierTo()

```
void VectSharp.IGraphicsContext.CubicBezierTo (
            double p1X,
            double p1Y,
            double p2X,
            double p2Y,
            double p3X,
            double p3Y )
```

Add to the current figure a cubic Bezier from the current point to a destination point, with two control points.

Parameters

p1X	The horizontal coordinate of the first control point.
p1Y	The vertical coordinate of the first control point.
p2X	The horizontal coordinate of the second control point.
p2Y	The vertical coordinate of the second control point.
p3X	The horizontal coordinate of the destination point.
p3Y	The vertical coordinate of the destination point.

7.80.2.2 DrawFilteredGraphics()

```
void VectSharp.IGraphicsContext.DrawFilteredGraphics (
            Graphics graphics,
            IFilter filter )
```

Draws a Graphics object, applying the specified *filter* .

Parameters

graphics	The Graphics object to draw on the current Graphics object.
filter	An IFilter object, representing the filter to apply to the *graphics* object.

7.80.2.3 DrawRasterImage()

```
void VectSharp.IGraphicsContext.DrawRasterImage (
            int sourceX,
            int sourceY,
            int sourceWidth,
            int sourceHeight,
            double destinationX,
            double destinationY,
            double destinationWidth,
            double destinationHeight,
            RasterImage image )
```

Draw a raster image.

Parameters

sourceX	The horizontal coordinate of the top-left corner of the rectangle delimiting the source area of the image.
sourceY	The vertical coordinate of the top-left corner of the rectangle delimiting the source area of the image.
sourceWidth	The width of the rectangle delimiting the source area of the image.
sourceHeight	The height of the rectangle delimiting the source area of the image.
destinationX	The horizontal coordinate of the top-left corner of the rectangle delimiting the destination area of the image.
destinationY	The vertical coordinate of the top-left corner of the rectangle delimiting the destination area of the image.
destinationWidth	The width of the rectangle delimiting the destination area of the image.
destinationHeight	The height of the rectangle delimiting the destination area of the image.
image	The image to draw.

7.80.2.4 Fill()

```
void VectSharp.IGraphicsContext.Fill (
            FillRule fillRule )
```

Fill the current path using the current FillStyle.

Parameters

fillRule	The FillRule that determines which parts of the path are filled.

7.80.2.5 FillText()

```
void VectSharp.IGraphicsContext.FillText (
            string text,
```

```
            double x,
            double y )
```

Fill a text string using the current Font and TextBaseline.

Parameters

text	The string to draw.
x	The horizontal coordinate of the text origin.
y	The vertical coordinate of the text origin.

7.80.2.6 LineTo()

```
void VectSharp.IGraphicsContext.LineTo (
            double x,
            double y )
```

Draw a line from the previous point to the specified point.

Parameters

x	The horizontal coordinate of the point.
y	The vertical coordinate of the point.

7.80.2.7 MoveTo()

```
void VectSharp.IGraphicsContext.MoveTo (
            double x,
            double y )
```

Change the current point without drawing a line from the previous point. If necessary, start a new figure.

Parameters

x	The horizontal coordinate of the point.
y	The vertical coordinate of the point.

7.80.2.8 Rectangle()

```
void VectSharp.IGraphicsContext.Rectangle (
            double x0,
```

```
            double y0,
            double width,
            double height )
```

Add a rectangle figure to the current path.

Parameters

x0	The horizontal coordinate of the top-left corner of the rectangle.
y0	The vertical coordinate of the top-left corner of the rectangle.
width	The width of corner of the rectangle.
height	The height of corner of the rectangle.

7.80.2.9 Rotate()

```
void VectSharp.IGraphicsContext.Rotate (
            double angle )
```

Rotate the coordinate system around the origin.

Parameters

angle	The angle (in radians) by which to rotate the coordinate system.

7.80.2.10 Scale()

```
void VectSharp.IGraphicsContext.Scale (
            double scaleX,
            double scaleY )
```

Scale the coordinate system with respect to the origin.

Parameters

scaleX	The horizontal scale.
scaleY	The vertical scale.

7.80.2.11 SetFillStyle() [1/2]

```
void VectSharp.IGraphicsContext.SetFillStyle (
            (int r, int g, int b, double a) style )
```

Set the current FillStyle.

Parameters

style	A ValueTuple<Int32, Int32, Int32, Double> containing component information for the colour. For r, g, and b, range: [0, 255]; for a, range: [0, 1].

7.80.2.12 SetFillStyle() [2/2]

```
void VectSharp.IGraphicsContext.SetFillStyle (
            Brush style )
```

Set the current FillStyle.

Parameters

style	The new fill style.

7.80.2.13 SetLineDash()

```
void VectSharp.IGraphicsContext.SetLineDash (
            LineDash dash )
```

Set the current line dash pattern.

Parameters

dash	The line dash pattern.

7.80.2.14 SetStrokeStyle() [1/2]

```
void VectSharp.IGraphicsContext.SetStrokeStyle (
            (int r, int g, int b, double a) style )
```

Set the current StrokeStyle.

Parameters

style	A ValueTuple<Int32, Int32, Int32, Double> containing component information for the colour. For r, g, and b, range: [0, 255]; for a, range: [0, 1].

7.80.2.15 SetStrokeStyle() [2/2]

```
void VectSharp.IGraphicsContext.SetStrokeStyle (
            Brush style )
```

Set the current StrokeStyle.

Parameters

style	The new stroke style.

7.80.2.16 StrokeText()

```
void VectSharp.IGraphicsContext.StrokeText (
            string text,
            double x,
            double y )
```

Stroke the outline of a text string using the current Font and TextBaseline.

Parameters

text	The string to draw.
x	The horizontal coordinate of the text origin.
y	The vertical coordinate of the text origin.

7.80.2.17 Transform()

```
void VectSharp.IGraphicsContext.Transform (
            double a,
            double b,
            double c,
            double d,
            double e,
            double f )
```

Transform the coordinate system with the specified transformation matrix [[a, c, e], [b, d, f], [0, 0, 1]].

Parameters

a	The first element of the first column.
b	The second element of the first column.
c	The first element of the second column.
d	The second element of the second column.
e	The first element of the third column.
f	The second element of the third column.

7.80.2.18 Translate()

```
void VectSharp.IGraphicsContext.Translate (
            double x,
            double y )
```

Translate the coordinate system origin.

Parameters

x	The horizontal translation.
y	The vertical translation.

7.80.3 Property Documentation

7.80.3.1 FillStyle

```
Brush VectSharp.IGraphicsContext.FillStyle  [get]
```

Current brush used to fill paths.

Definition at line 147 of file Graphics.cs.

7.80.3.2 Font

```
Font VectSharp.IGraphicsContext.Font  [get], [set]
```

The current font.

Definition at line 92 of file Graphics.cs.

7.80.3.3 Height

```
double VectSharp.IGraphicsContext.Height  [get]
```

Height of the graphic surface.

Definition at line 46 of file Graphics.cs.

7.80.3.4 LineCap

```
LineCaps VectSharp.IGraphicsContext.LineCap  [set]
```

Current line cap used to stroke paths.

Definition at line 212 of file Graphics.cs.

7.80.3.5 LineJoin

```
LineJoins VectSharp.IGraphicsContext.LineJoin  [set]
```

Current line join used to stroke paths.

Definition at line 217 of file Graphics.cs.

7.80.3.6 LineWidth

```
double VectSharp.IGraphicsContext.LineWidth  [get], [set]
```

Current line width used to stroke paths.

Definition at line 207 of file Graphics.cs.

7.80.3.7 StrokeStyle

```
Brush VectSharp.IGraphicsContext.StrokeStyle  [get]
```

Current brush used to stroke paths.

Definition at line 164 of file Graphics.cs.

7.80.3.8 Tag

```
string VectSharp.IGraphicsContext.Tag  [get], [set]
```

The current tag. How this can be used depends on each implementation.

Definition at line 228 of file Graphics.cs.

7.80.3.9 TextBaseline

```
TextBaselines VectSharp.IGraphicsContext.TextBaseline  [get], [set]
```

The current text baseline.

Definition at line 97 of file Graphics.cs.

7.80.3.10 Width

```
double VectSharp.IGraphicsContext.Width  [get]
```

Width of the graphic surface.

Definition at line 41 of file Graphics.cs.

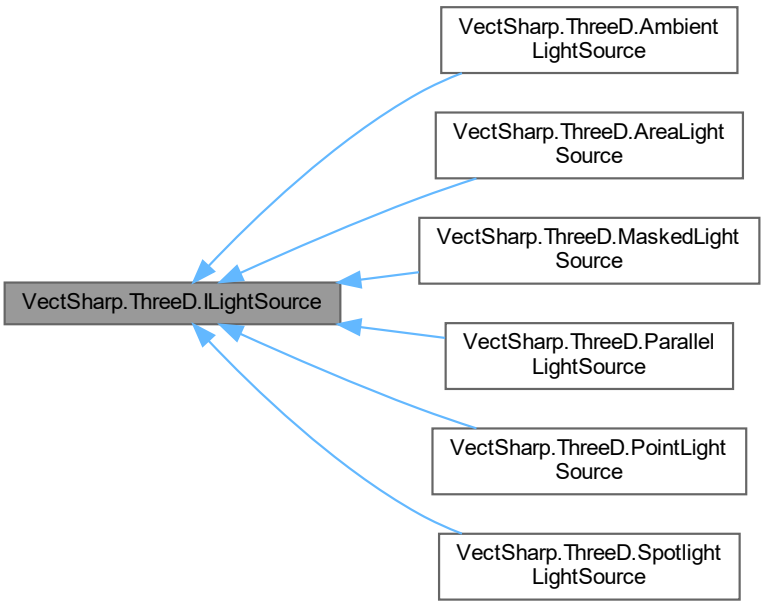
The documentation for this interface was generated from the following file:

- VectSharp/Graphics.cs

7.81 VectSharp.ThreeD.ILightSource Interface Reference

Represents a light source.

Inheritance diagram for VectSharp.ThreeD.ILightSource:



Public Member Functions

- LightIntensity GetLightAt (Point3D point)

  Computes the light intensity at the specified point, without taking into account any obstructions.
- double GetObstruction (Point3D point, IEnumerable< Triangle3DElement > shadowingTriangles)

  Determines the amount of obstruction of the light that results at point due to the specified shadowingTriangles .

Properties

- bool CastsShadow `[get]`

  Determines whether the light casts a shadow or not.

7.81.1 Detailed Description

Represents a light source.

Definition at line 65 of file Lights.cs.

7.81.2 Member Function Documentation

7.81.2.1 GetLightAt()

```
LightIntensity VectSharp.ThreeD.ILightSource.GetLightAt (
            Point3D point )
```

Computes the light intensity at the specified point, without taking into account any obstructions.

Parameters

point	The Point3DElement at which the light intensity should be computed.

Returns

Implemented in VectSharp.ThreeD.AmbientLightSource, VectSharp.ThreeD.ParallelLightSource, VectSharp.ThreeD.PointLightSource, VectSharp.ThreeD.SpotlightLightSource, VectSharp.ThreeD.MaskedLightSource, and VectSharp.ThreeD.AreaLightSource.

7.81.2.2 GetObstruction()

```
double VectSharp.ThreeD.ILightSource.GetObstruction (
            Point3D point,
            IEnumerable< Triangle3DElement > shadowingTriangles )
```

Determines the amount of obstruction of the light that results at *point* due to the specified *shadowingTriangles* .

Parameters

point	The Point3D at which the obstruction should be computed.
shadowingTriangles	A collection of Triangle3DElement casting shadows.

Returns

1 if the light is completely obstructed, 0 if the light is completely visible, a value between these if the light is partially obstructed.

Implemented in VectSharp.ThreeD.AmbientLightSource, VectSharp.ThreeD.ParallelLightSource, VectSharp.ThreeD.PointLightSource, VectSharp.ThreeD.SpotlightLightSource, VectSharp.ThreeD.MaskedLightSource, and VectSharp.ThreeD.AreaLightSource.

7.81.3 Property Documentation

7.81.3.1 CastsShadow

```
bool VectSharp.ThreeD.ILightSource.CastsShadow  [get]
```

Determines whether the light casts a shadow or not.

Implemented in VectSharp.ThreeD.AmbientLightSource, VectSharp.ThreeD.ParallelLightSource, VectSharp.ThreeD.PointLightSource, VectSharp.ThreeD.SpotlightLightSource, VectSharp.ThreeD.MaskedLightSource, and VectSharp.ThreeD.AreaLightSource.

Definition at line 77 of file Lights.cs.

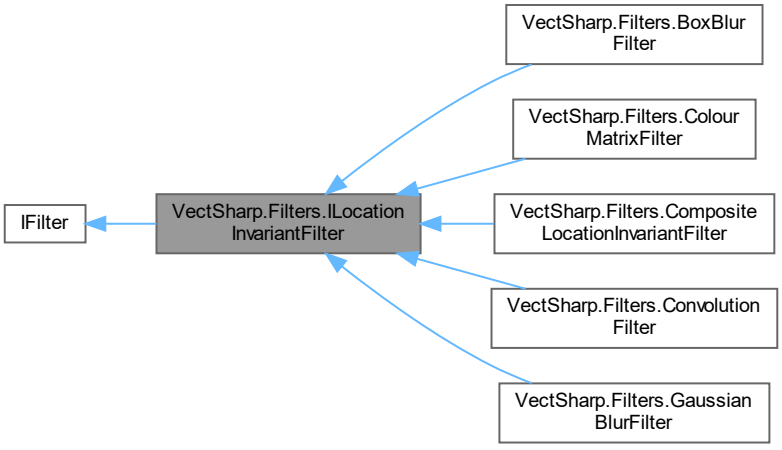
The documentation for this interface was generated from the following file:

- VectSharp.ThreeD/Lights.cs

7.82 VectSharp.Filters.ILocationInvariantFilter Interface Reference

Represents a filter that can be applied to an image regardless of its location on the graphics surface.

Inheritance diagram for VectSharp.Filters.ILocationInvariantFilter:



Public Member Functions

- RasterImage Filter (RasterImage image, double scale)

  Applies the filter to a RasterImage.

Additional Inherited Members

7.82.1 Detailed Description

Represents a filter that can be applied to an image regardless of its location on the graphics surface.

Definition at line 42 of file Filters.cs.

7.82.2 Member Function Documentation

7.82.2.1 Filter()

```
RasterImage VectSharp.Filters.ILocationInvariantFilter.Filter (
            RasterImage image,
            double scale )
```

Applies the filter to a RasterImage.

Parameters

image	The RasterImage to which the filter will be applied.
scale	The scale of the image with respect to the filter.

Returns

A new RasterImage containing the filtered image. The source *image* is left unaltered.

Implemented in VectSharp.Filters.BoxBlurFilter, VectSharp.Filters.ColourMatrixFilter, VectSharp.Filters.CompositeLocationInvariantFil
VectSharp.Filters.ConvolutionFilter, and VectSharp.Filters.GaussianBlurFilter.

The documentation for this interface was generated from the following file:

- VectSharp/Filters/Filters.cs

7.83 VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter Class Reference

Contains methods to render a Page to an Image.

Classes

- class UnknownFormatException

  The exception that is raised when the output file format is not specified and the file name does not have an extension corresponding to a known file format.

Static Public Member Functions

- static Image< SixLabors.ImageSharp.PixelFormats.Rgba32 > SaveAsImage (this Page page, double scale=1)

  Render the page to an Image object.
- static void SaveAsImage (this Page page, Stream imageStream, OutputFormats outputFormat, double scale=1)

  Render the page to an image stream.
- static void SaveAsImage (this Page page, string fileName, OutputFormats? outputFormat=null, double scale=1)

  Render the page to an image file.
- static DisposableIntPtr SaveAsRawBytes (this Page pag, out int width, out int height, out int totalSize, double scale=1)

  Render the page to raw pixel data, in 32bpp RGBA format.
- static byte[ ] SaveAsRawBytes (this Page pag, out int width, out int height, double scale=1)

  Return the page to raw pixel data, in 32bpp RGBA format.
- static RasterImage Rasterise (this Graphics graphics, Rectangle region, double scale, bool interpolate)

  Rasterise a region of a Graphics object.
- static Image SaveAsAnimatedGIF (this Animation animation, double scale=1, double frameRate=50, double durationScaling=1, GifColorTableMode colorTableMode=GifColorTableMode.Local)

  Saves the animation to an animated GIF.
- static void SaveAsAnimatedGIF (this Animation animation, Stream imageStream, double scale=1, double frameRate=50, double durationScaling=1, GifColorTableMode colorTableMode=GifColorTableMode.Local)

  Saves the animation to an animated GIF stream.
- static void SaveAsAnimatedGIF (this Animation animation, string fileName, double scale=1, double frame↩ Rate=50, double durationScaling=1, GifColorTableMode colorTableMode=GifColorTableMode.Local)

  Saves the animation to an animated GIF file.
- static void SaveAsAnimatedPNG (this Animation animation, Stream imageStream, double scale=1, double frameRate=60, double durationScaling=1, AnimatedPNG.InterframeCompression interframe↩ Compression=AnimatedPNG.InterframeCompression.First)

  Saves the animation to a stream in animated PNG format.
- static void SaveAsAnimatedPNG (this Animation animation, string fileName, double scale=1, double frameRate=60, double durationScaling=1, AnimatedPNG.InterframeCompression interframe↩ Compression=AnimatedPNG.InterframeCompression.First)

  Saves the animation to an animated PNG file.

7.83.1 Detailed Description

Contains methods to render a Page to an Image.

Definition at line 1048 of file ImageSharpContext.cs.

7.83.2 Member Function Documentation

7.83.2.1 Rasterise()

```
static RasterImage VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.Rasterise (
            this Graphics graphics,
            Rectangle region,
            double scale,
            bool interpolate )  [static]
```

Rasterise a region of a Graphics object.

Parameters

graphics	The Graphics object that will be rasterised.
region	The region of the *graphics* that will be rasterised.
scale	The scale at which the image will be rendered.
interpolate	Whether the resulting image should be interpolated or not when it is drawn on another Graphics surface.

Returns

A RasterImage containing the rasterised graphics.

Definition at line 1280 of file ImageSharpContext.cs.

7.83.2.2 SaveAsAnimatedGIF() [1/3]

```
static Image VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.SaveAsAnimatedGIF (
            this Animation animation,
            double scale = 1,
            double frameRate = 50,
            double durationScaling = 1,
            GifColorTableMode colorTableMode = GifColorTableMode.Local )  [static]
```

Saves the animation to an animated GIF.

Parameters

animation	The animation to export.
scale	The scale at which the animation will be rendered.
frameRate	The target frame rate of the animation, in frames-per-second (fps). This is capped by the animated GIF specification at 50 fps.
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
colorTableMode	Determines whether a single colour table should be used for the whole image, or if a different colour table should be used for each frame.

Returns

An Image containing the animated GIF.

Definition at line 1328 of file ImageSharpContext.cs.

7.83.2.3 SaveAsAnimatedGIF() [2/3]

```
static void VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.SaveAsAnimatedGIF (
            this Animation animation,
            Stream imageStream,
            double scale = 1,
            double frameRate = 50,
            double durationScaling = 1,
            GifColorTableMode colorTableMode = GifColorTableMode.Local )  [static]
```

Saves the animation to an animated GIF stream.

Parameters

animation	The animation to export.
imageStream	The stream on which the animated GIF will be written.
scale	The scale at which the animation will be rendered.
frameRate	The target frame rate of the animation, in frames-per-second (fps). This is capped by the animated GIF specification at 50 fps.
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
colorTableMode	Determines whether a single colour table should be used for the whole image, or if a different colour table should be used for each frame.

Definition at line 1387 of file ImageSharpContext.cs.

7.83.2.4 SaveAsAnimatedGIF() [3/3]

```
static void VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.SaveAsAnimatedGIF (
            this Animation animation,
            string fileName,
            double scale = 1,
            double frameRate = 50,
            double durationScaling = 1,
            GifColorTableMode colorTableMode = GifColorTableMode.Local )  [static]
```

Saves the animation to an animated GIF file.

Parameters

animation	The animation to export.
fileName	The output file to create.

Parameters

scale	The scale at which the animation will be rendered.
frameRate	The target frame rate of the animation, in frames-per-second (fps). This is capped by the animated GIF specification at 50 fps.
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
colorTableMode	Determines whether a single colour table should be used for the whole image, or if a different colour table should be used for each frame.

Definition at line 1403 of file ImageSharpContext.cs.

7.83.2.5 SaveAsAnimatedPNG() [1/2]

```
static void VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.SaveAsAnimatedPNG (
            this Animation animation,
            Stream imageStream,
            double scale = 1,
            double frameRate = 60,
            double durationScaling = 1,
            AnimatedPNG.InterframeCompression interframeCompression = AnimatedPNG.InterframeCompression.First
)  [static]
```

Saves the animation to a stream in animated PNG format.

Parameters

animation	The animation to export.
imageStream	The stream on which the animated PNG will be written.
scale	The scale at which the animation will be rendered.
frameRate	The target frame rate of the animation, in frames-per-second (fps). This is capped by the animated PNG specification at 90 fps.
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
interframeCompression	The kind of compression that will be used to reduce file size. Note that if the animation has a transparent background, no compression can be performed, and the value of this parameter is ignored.

Definition at line 1420 of file ImageSharpContext.cs.

7.83.2.6 SaveAsAnimatedPNG() [2/2]

```
static void VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.SaveAsAnimatedPNG (
            this Animation animation,
            string fileName,
```

```
            double scale = 1,
            double frameRate = 60,
            double durationScaling = 1,
            AnimatedPNG.InterframeCompression interframeCompression = AnimatedPNG.InterframeCompression.First
)  [static]
```

Saves the animation to an animated PNG file.

Parameters

animation	The animation to export.
fileName	The output file to create.
scale	The scale at which the animation will be rendered.
frameRate	The target frame rate of the animation, in frames-per-second (fps). This is capped by the animated PNG specification at 90 fps.
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
interframeCompression	The kind of compression that will be used to reduce file size. Note that if the animation has a transparent background, no compression can be performed, and the value of this parameter is ignored.

Definition at line 1508 of file ImageSharpContext.cs.

7.83.2.7 SaveAsImage() [1/3]

```
static Image< SixLabors.ImageSharp.PixelFormats.Rgba32 > VectSharp.Raster.ImageSharp.Image↩
SharpContextInterpreter.SaveAsImage (
            this Page page,
            double scale = 1 )  [static]
```

Render the page to an Image object.

Parameters

page	The Page to render.
scale	The scale to be used when rasterising the page. This will determine the width and height of the Image.

Returns

An Image containing the rasterised page.

Definition at line 1056 of file ImageSharpContext.cs.

7.83.2.8 SaveAsImage() [2/3]

```
static void VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.SaveAsImage (
            this Page page,
```

```
            Stream imageStream,
            OutputFormats outputFormat,
            double scale = 1 )  [static]
```

Render the page to an image stream.

Parameters

page	The Page to render.
imageStream	The Stream on which the image data will be written.
outputFormat	The format of the image that will be created.
scale	The scale to be used when rasterising the page. This will determine the width and height of the image.

Definition at line 1072 of file ImageSharpContext.cs.

7.83.2.9 SaveAsImage() [3/3]

```
static void VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.SaveAsImage (
            this Page page,
            string fileName,
            OutputFormats?  outputFormat = null,
            double scale = 1 )  [static]
```

Render the page to an image file.

Parameters

page	The Page to render.
fileName	The path of the file where the image will be saved.
outputFormat	The format of the image that will be created. If this is `null` (the default), the format is desumed from the extension of the file.
scale	The scale to be used when rasterising the page. This will determine the width and height of the image.

Definition at line 1130 of file ImageSharpContext.cs.

7.83.2.10 SaveAsRawBytes() [1/2]

```
static byte[] VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.SaveAsRawBytes (
            this Page pag,
            out int width,
            out int height,
            double scale = 1 )  [static]
```

Return the page to raw pixel data, in 32bpp RGBA format.

Parameters

pag	The Page to render.
scale	The scale to be used when rasterising the page. This will determine the width and height of the image.
width	The width of the rendered image.
height	The height of the rendered image.

Returns

A byte array containing the raw pixel data.

Definition at line 1243 of file ImageSharpContext.cs.

7.83.2.11 SaveAsRawBytes() [2/2]

```
static DisposableIntPtr VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.SaveAsRaw↩
Bytes (
            this Page pag,
            out int width,
            out int height,
            out int totalSize,
            double scale = 1 )  [static]
```

Render the page to raw pixel data, in 32bpp RGBA format.

Parameters

pag	The Page to render.
scale	The scale to be used when rasterising the page. This will determine the width and height of the image.
width	The width of the rendered image.
height	The height of the rendered image.
totalSize	The size in bytes of the raw pixel data.

Returns

A DisposableIntPtr containing a pointer to the raw pixel data, stored in unmanaged memory. Dispose this object to release the unmanaged memory.

Definition at line 1200 of file ImageSharpContext.cs.

The documentation for this class was generated from the following file:

- VectSharp.Raster.ImageSharp/ImageSharpContext.cs

7.84 VectSharp.ImageSharpUtils.ImageURIParser Class Reference

Provides a method to parse an image URI into a page.

Static Public Member Functions

- static Func< string, bool, Page > Parser (Func< string, bool, Page > parseSVG)

  Parses an image URI into a page. This is intended to replace the default image URI interpreter in `VectSharp.SVG.Parser.ParseImageURI`. To do this, use something like:

7.84.1 Detailed Description

Provides a method to parse an image URI into a page.

Definition at line 29 of file ImageUriParser.cs.

7.84.2 Member Function Documentation

7.84.2.1 Parser()

```
static Func< string, bool, Page > VectSharp.ImageSharpUtils.ImageURIParser.Parser (
            Func< string, bool, Page > parseSVG )  [static]
```

Parses an image URI into a page. This is intended to replace the default image URI interpreter in `VectSharp.SVG.Parser.ParseImageURI`. To do this, use something like:

```
VectSharp.SVG.Parser.ParseImageURI = VectSharp.ImageSharpUtils.ImageURIParser.Parser(Vec
```

Parameters

parseSVG	A function to parse an SVG image uri into a page. You should pass `VectSharp.SVG.Parser.ParseSVGURI` as this argument.

Returns

A function to parse an image URI into a page.

Definition at line 37 of file ImageUriParser.cs.

The documentation for this class was generated from the following file:

- VectSharp.ImageSharpUtils/ImageUriParser.cs

7.85 VectSharp.MuPDFUtils.ImageURIParser Class Reference

Provides a method to parse an image URI into a page.

Static Public Member Functions

- static Func< string, bool, Page > Parser (Func< string, bool, Page > parseSVG)

  Parses an image URI into a page. This is intended to replace the default image URI interpreter in `VectSharp.SVG.Parser.ParseImageURI`. To do this, use something like:

7.85.1 Detailed Description

Provides a method to parse an image URI into a page.

Definition at line 29 of file ImageURIParser.cs.

7.85.2 Member Function Documentation

7.85.2.1 Parser()

```
static Func< string, bool, Page > VectSharp.MuPDFUtils.ImageURIParser.Parser (
            Func< string, bool, Page > parseSVG )  [static]
```

Parses an image URI into a page. This is intended to replace the default image URI interpreter in `VectSharp.SVG.Parser.ParseImageURI`. To do this, use something like:

```
VectSharp.SVG.Parser.ParseImageURI = VectSharp.MuPDFUtils.ImageURIParser.Parser(VectShar
```

Parameters

parseSVG	A function to parse an SVG image uri into a page. You should pass `VectSharp.SVG.Parser.ParseSVGURI` as this argument.

Returns

A function to parse an image URI into a page.

Definition at line 37 of file ImageURIParser.cs.

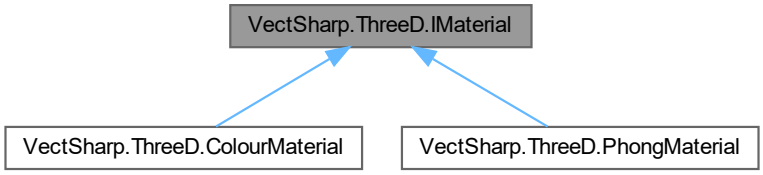
The documentation for this class was generated from the following file:

- VectSharp.MuPDFUtils/ImageURIParser.cs

7.86 VectSharp.ThreeD.IMaterial Interface Reference

Represents a material used to the determine the appearance of Triangle3DElement.

Inheritance diagram for VectSharp.ThreeD.IMaterial:



Public Member Functions

- Colour GetColour (Point3D point, NormalizedVector3D surfaceNormal, Camera camera, IList< ILightSource > lights, IList< double > obstructions)

  Obtains the Colour at the specified point.

7.86.1 Detailed Description

Represents a material used to the determine the appearance of Triangle3DElement.

Definition at line 31 of file Materials.cs.

7.86.2 Member Function Documentation

7.86.2.1 GetColour()

```
Colour VectSharp.ThreeD.IMaterial.GetColour (
            Point3D point,
            NormalizedVector3D surfaceNormal,
            Camera camera,
            IList< ILightSource > lights,
            IList< double > obstructions )
```

Obtains the Colour at the specified point.

Parameters

point	The point whose colour should be determined.
surfaceNormal	The normal to the surface at the specified *point* .
camera	The camera being used to render the scene.
lights	A list of light sources that are present in the scene.
obstructions	A list of values indicating how obstructed each light source is.

Returns

The Colour of the specified point.

Implemented in VectSharp.ThreeD.ColourMaterial, and VectSharp.ThreeD.PhongMaterial.

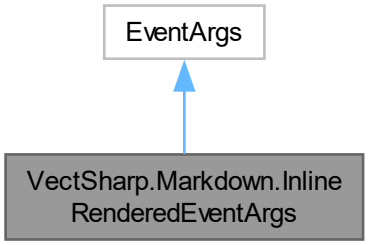
The documentation for this interface was generated from the following file:

- VectSharp.ThreeD/Materials.cs

7.87 VectSharp.Markdown.InlineRenderedEventArgs Class Reference

EventArgs for the MarkdownRenderer.InlineRendered event.

Inheritance diagram for VectSharp.Markdown.InlineRenderedEventArgs:



Public Member Functions

- InlineRenderedEventArgs (MarkdownContext markdownContext, Graphics graphics, Inline inline)

  Create a new InlineRenderingEventArgs instance.

Properties

- MarkdownContext MarkdownContext `[get, set]`

  The current VectSharp.Markdown.MarkdownContext.
- Graphics Graphics `[get, set]`

  The current VectSharp.Graphics surface.
- Inline Inline `[get]`

  The Markdig.Syntax.Inlines.Inline that has been rendered.

7.87.1 Detailed Description

EventArgs for the MarkdownRenderer.InlineRendered event.

Definition at line 407 of file MarkdownRenderer.Events.cs.

7.87.2 Constructor & Destructor Documentation

7.87.2.1 InlineRenderedEventArgs()

```
VectSharp.Markdown.InlineRenderedEventArgs.InlineRenderedEventArgs (
            MarkdownContext markdownContext,
            Graphics graphics,
            Inline inline )
```

Create a new InlineRenderingEventArgs instance.

Parameters

markdownContext	The current VectSharp.Markdown.MarkdownContext.
graphics	The current VectSharp.Graphics surface.
inline	The Markdig.Syntax.Inlines.Inline that has been rendered.

Definition at line 430 of file MarkdownRenderer.Events.cs.

7.87.3 Property Documentation

7.87.3.1 Graphics

```
Graphics VectSharp.Markdown.InlineRenderedEventArgs.Graphics  [get], [set]
```

The current VectSharp.Graphics surface.

Definition at line 417 of file MarkdownRenderer.Events.cs.

7.87.3.2 Inline

```
Inline VectSharp.Markdown.InlineRenderedEventArgs.Inline  [get]
```

The Markdig.Syntax.Inlines.Inline that has been rendered.

Definition at line 422 of file MarkdownRenderer.Events.cs.

7.87.3.3 MarkdownContext

```
MarkdownContext VectSharp.Markdown.InlineRenderedEventArgs.MarkdownContext  [get], [set]
```

The current VectSharp.Markdown.MarkdownContext.

Definition at line 412 of file MarkdownRenderer.Events.cs.

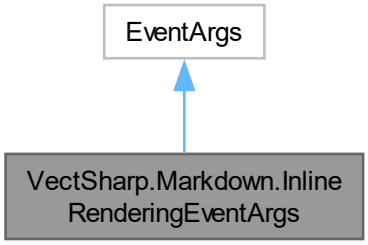
The documentation for this class was generated from the following file:

- VectSharp.Markdown/MarkdownRenderer.Events.cs

7.88 VectSharp.Markdown.InlineRenderingEventArgs Class Reference

EventArgs for the MarkdownRenderer.InlineRendering event.

Inheritance diagram for VectSharp.Markdown.InlineRenderingEventArgs:



Public Member Functions

- InlineRenderingEventArgs (MarkdownContext markdownContext, Graphics graphics, Inline inline)

  Create a new InlineRenderingEventArgs instance.

Properties

- MarkdownContext MarkdownContext `[get, set]`

  The current VectSharp.Markdown.MarkdownContext.
- Graphics Graphics `[get, set]`

  The VectSharp.Graphics surface on which the Inline will be drawn.
- Inline Inline `[get, set]`

  The Markdig.Syntax.Inlines.Inline that will be rendered.

7.88.1 Detailed Description

EventArgs for the MarkdownRenderer.InlineRendering event.

Definition at line 373 of file MarkdownRenderer.Events.cs.

7.88.2 Constructor & Destructor Documentation

7.88.2.1 InlineRenderingEventArgs()

```
VectSharp.Markdown.InlineRenderingEventArgs.InlineRenderingEventArgs (
            MarkdownContext markdownContext,
            Graphics graphics,
            Inline inline )
```

Create a new InlineRenderingEventArgs instance.

Parameters

markdownContext	The current VectSharp.Markdown.MarkdownContext.
graphics	The VectSharp.Graphics surface on which the Inline will be drawn.
inline	The Markdig.Syntax.Inlines.Inline that will be rendered.

Definition at line 396 of file MarkdownRenderer.Events.cs.

7.88.3 Property Documentation

7.88.3.1 Graphics

```
Graphics VectSharp.Markdown.InlineRenderingEventArgs.Graphics  [get], [set]
```

The VectSharp.Graphics surface on which the Inline will be drawn.

Definition at line 383 of file MarkdownRenderer.Events.cs.

7.88.3.2 Inline

```
Inline VectSharp.Markdown.InlineRenderingEventArgs.Inline  [get], [set]
```

The Markdig.Syntax.Inlines.Inline that will be rendered.

Definition at line 388 of file MarkdownRenderer.Events.cs.

7.88.3.3 MarkdownContext

```
MarkdownContext VectSharp.Markdown.InlineRenderingEventArgs.MarkdownContext  [get], [set]
```

The current VectSharp.Markdown.MarkdownContext.

Definition at line 378 of file MarkdownRenderer.Events.cs.

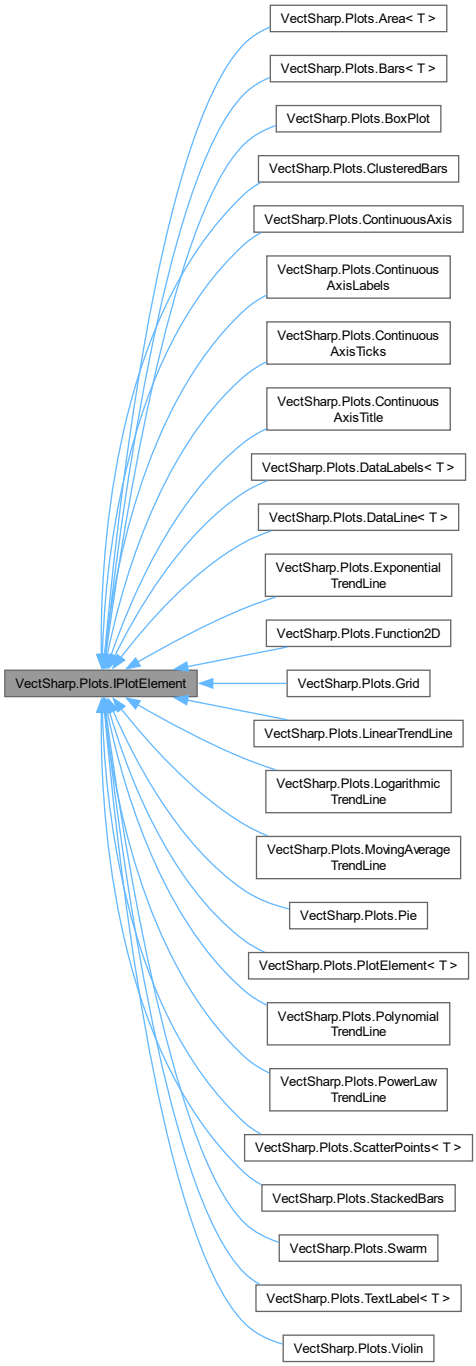
The documentation for this class was generated from the following file:

- VectSharp.Markdown/MarkdownRenderer.Events.cs

7.89 VectSharp.Plots.IPlotElement Interface Reference

Represents a plot element.

Inheritance diagram for VectSharp.Plots.IPlotElement:



Public Member Functions

- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Properties

- ICoordinateSystem CoordinateSystem [get]

  The coordinate system used to transform the points from data space to plot space.

7.89.1 Detailed Description

Represents a plot element.

Definition at line 91 of file Plot.cs.

7.89.2 Member Function Documentation

7.89.2.1 Plot()

```
void VectSharp.Plots.IPlotElement.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implemented in VectSharp.Plots.ContinuousAxis, VectSharp.Plots.ContinuousAxisTicks, VectSharp.Plots.ContinuousAxisLabels, VectSharp.Plots.ContinuousAxisTitle, VectSharp.Plots.Grid, VectSharp.Plots.Bars< T >, VectSharp.Plots.StackedBars, VectSharp.Plots.ClusteredBars, VectSharp.Plots.BoxPlot, VectSharp.Plots.ScatterPoints< T >, VectSharp.Plots.TextLabel< T >, VectSharp.Plots.DataLabels< T >, VectSharp.Plots.DataLine< T >, VectSharp.Plots.Area< T >, VectSharp.Plots.Function2D, VectSharp.Plots.Pie, VectSharp.Plots.PlotElement< T >, VectSharp.Plots.Swarm, VectSharp.Plots.LinearTrendLine, VectSharp.Plots.ExponentialTrendLine, VectSharp.Plots.LogarithmicTrendLine, VectSharp.Plots.PolynomialTrendLine, VectSharp.Plots.PowerLawTrendLine, VectSharp.Plots.MovingAverageTrendLine, and VectSharp.Plots.Violin.

7.89.3 Property Documentation

7.89.3.1 CoordinateSystem

```
ICoordinateSystem VectSharp.Plots.IPlotElement.CoordinateSystem [get]
```

The coordinate system used to transform the points from data space to plot space.

Implemented in VectSharp.Plots.ContinuousAxis, VectSharp.Plots.ContinuousAxisTicks, VectSharp.Plots.ContinuousAxisLabels, VectSharp.Plots.ContinuousAxisTitle, VectSharp.Plots.Grid, VectSharp.Plots.Bars< T >, VectSharp.Plots.StackedBars, VectSharp.Plots.ClusteredBars, VectSharp.Plots.BoxPlot, VectSharp.Plots.ScatterPoints< T >, VectSharp.Plots.TextLabel< T >,

VectSharp.Plots.DataLabels< T >, VectSharp.Plots.DataLine< T >, VectSharp.Plots.Area< T >, VectSharp.Plots.Function2D, VectSharp.Plots.Pie, VectSharp.Plots.PlotElement< T >, VectSharp.Plots.Swarm, VectSharp.Plots.LinearTrendLine, VectSharp.Plots.ExponentialTrendLine, VectSharp.Plots.LogarithmicTrendLine, VectSharp.Plots.PolynomialTrendLine, VectSharp.Plots.PowerLawTrendLine, VectSharp.Plots.MovingAverageTrendLine, and VectSharp.Plots.Violin.

Definition at line 102 of file Plot.cs.

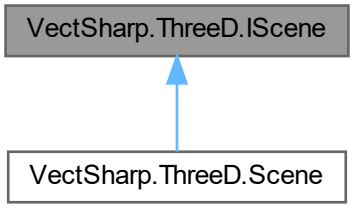
The documentation for this interface was generated from the following file:

- VectSharp.Plots/Plot.cs

7.90 VectSharp.ThreeD.IScene Interface Reference

Represents a 3D scene.

Inheritance diagram for VectSharp.ThreeD.IScene:



Public Member Functions

- void AddElement (Element3D element)

  Adds the specified element to the scene.
- void AddRange (IEnumerable< Element3D > elements)

  Adds the specified elements to the scene.
- void Replace (Func< Element3D, Element3D > replacementFunction)

  Replaces each element in the scene with the element returned by the replacementFunction .
- void Replace (Func< Element3D, IEnumerable< Element3D > > replacementFunction)

  Replaces each element in the scene with the element(s) returned by the replacementFunction .

Properties

- IEnumerable< Element3D > SceneElements `[get]`

  The Element3Ds constituting the scene.
- object SceneLock `[get]`

  An object used to synchronise multithreaded rendering of the same scene.

7.90.1 Detailed Description

Represents a 3D scene.

Definition at line 26 of file Scene.cs.

7.90.2 Member Function Documentation

7.90.2.1 AddElement()

```
void VectSharp.ThreeD.IScene.AddElement (
            Element3D element )
```

Adds the specified *element* to the scene.

Parameters

element	The Element3D to add.

Implemented in VectSharp.ThreeD.Scene.

7.90.2.2 AddRange()

```
void VectSharp.ThreeD.IScene.AddRange (
            IEnumerable< Element3D > elements )
```

Adds the specified *elements* to the scene.

Parameters

elements	A collection of Element3Ds to add.

Implemented in VectSharp.ThreeD.Scene.

7.90.2.3 Replace() [1/2]

```
void VectSharp.ThreeD.IScene.Replace (
            Func< Element3D, Element3D > replacementFunction )
```

Replaces each element in the scene with the element returned by the *replacementFunction* .

Parameters

replacementFunction	A function replacing each Element3D in the scene with another Element3D.

Implemented in VectSharp.ThreeD.Scene.

7.90.2.4 Replace() [2/2]

```
void VectSharp.ThreeD.IScene.Replace (
            Func< Element3D, IEnumerable< Element3D > > replacementFunction )
```

Replaces each element in the scene with the element(s) returned by the *replacementFunction* .

Parameters

replacementFunction	A function replacing each Element3D in the scene with 0 or more Element3Ds.

Implemented in VectSharp.ThreeD.Scene.

7.90.3 Property Documentation

7.90.3.1 SceneElements

```
IEnumerable<Element3D> VectSharp.ThreeD.IScene.SceneElements  [get]
```

The Element3Ds constituting the scene.

Implemented in VectSharp.ThreeD.Scene.

Definition at line 31 of file Scene.cs.

7.90.3.2 SceneLock

```
object VectSharp.ThreeD.IScene.SceneLock  [get]
```

An object used to synchronise multithreaded rendering of the same scene.

Implemented in VectSharp.ThreeD.Scene.

Definition at line 60 of file Scene.cs.

The documentation for this interface was generated from the following file:

- VectSharp.ThreeD/Scene.cs

7.91 VectSharp.ThreeD.LightIntensity Struct Reference

Represents the intensity of a light source at a particular point.

Public Member Functions

- LightIntensity (double intensity, NormalizedVector3D direction)

  Creates a new LightIntensity.
- void Deconstruct (out double intensity, out NormalizedVector3D direction)

  Deconstructs the struct.

Public Attributes

- double Intensity

  The intensity of the light.
- NormalizedVector3D Direction

  The direction towards from which the light comes.

7.91.1 Detailed Description

Represents the intensity of a light source at a particular point.

Definition at line 27 of file Lights.cs.

7.91.2 Constructor & Destructor Documentation

7.91.2.1 LightIntensity()

```
VectSharp.ThreeD.LightIntensity.LightIntensity (
            double intensity,
            NormalizedVector3D direction )
```

Creates a new LightIntensity.

Parameters

intensity	The intensity of the light.
direction	The direction from which the light comes.

Definition at line 44 of file Lights.cs.

7.91.3 Member Function Documentation

7.91.3.1 Deconstruct()

```
void VectSharp.ThreeD.LightIntensity.Deconstruct (
            out double intensity,
            out NormalizedVector3D direction )
```

Deconstructs the struct.

Parameters

intensity	This parameter will hold the Intensity of the light.
direction	This parameter will hold the Direction of the light.

Definition at line 55 of file Lights.cs.

7.91.4 Member Data Documentation

7.91.4.1 Direction

```
NormalizedVector3D VectSharp.ThreeD.LightIntensity.Direction
```

The direction towards from which the light comes.

Definition at line 37 of file Lights.cs.

7.91.4.2 Intensity

```
double VectSharp.ThreeD.LightIntensity.Intensity
```

The intensity of the light.

Definition at line 32 of file Lights.cs.

The documentation for this struct was generated from the following file:

- VectSharp.ThreeD/Lights.cs

7.92 VectSharp.Markdown.Line Class Reference

Represents a text line.

Public Member Functions

- Line (double initialAscent)

  Create a new Line.

Properties

- List< LineFragment > Fragments [get]

  The fragments contained in the line.
- double InitialAscent [get]

  The initial ascent for text on the Line.

7.92.1 Detailed Description

Represents a text line.

Definition at line 354 of file Line.cs.

7.92.2 Constructor & Destructor Documentation

7.92.2.1 Line()

```
VectSharp.Markdown.Line.Line (
            double initialAscent )
```

Create a new Line.

Parameters

initialAscent	The initial ascent for text on the Line.

Definition at line 370 of file Line.cs.

7.92.3 Property Documentation

7.92.3.1 Fragments

```
List<LineFragment> VectSharp.Markdown.Line.Fragments  [get]
```

The fragments contained in the line.

Definition at line 359 of file Line.cs.

7.92.3.2 InitialAscent

```
double VectSharp.Markdown.Line.InitialAscent  [get]
```

The initial ascent for text on the Line.

Definition at line 364 of file Line.cs.

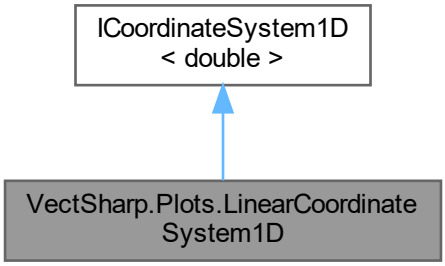
The documentation for this class was generated from the following file:

- VectSharp.Markdown/Line.cs

7.93 VectSharp.Plots.LinearCoordinateSystem1D Class Reference

Represents a 1-D linear coordinate system.

Inheritance diagram for VectSharp.Plots.LinearCoordinateSystem1D:



Public Member Functions

- LinearCoordinateSystem1D (double min, double max, double scale)

  Creates a new LinearCoordinateSystem1D, manually specifying the parameters.
- LinearCoordinateSystem1D (IReadOnlyList< double > data, double scale=350)

  Creates a new LinearCoordinateSystem1D determining the value range from the data .
- double ToPlotCoordinates (double dataPoint)

Properties

- double Min [get, set]

  The minimum value.
- double Max [get, set]

  The maximum value.
- double Scale [get, set]

  The scale factor.

7.93.1 Detailed Description

Represents a 1-D linear coordinate system.

Definition at line 924 of file CoordinateSystems.cs.

7.93.2 Constructor & Destructor Documentation

7.93.2.1 LinearCoordinateSystem1D() [1/2]

```
VectSharp.Plots.LinearCoordinateSystem1D.LinearCoordinateSystem1D (
            double min,
            double max,
            double scale )
```

Creates a new LinearCoordinateSystem1D, manually specifying the parameters.

Parameters

min	The minimum value.
max	The maximum value.
scale	The scale factor.

Definition at line 947 of file CoordinateSystems.cs.

7.93.2.2 LinearCoordinateSystem1D() [2/2]

```
VectSharp.Plots.LinearCoordinateSystem1D.LinearCoordinateSystem1D (
            IReadOnlyList< double > data,
            double scale = 350 )
```

Creates a new LinearCoordinateSystem1D determining the value range from the *data* .

Parameters

data	The data from which the value range should be determined.
scale	The scale factor.

Definition at line 959 of file CoordinateSystems.cs.

7.93.3 Member Function Documentation

7.93.3.1 ToPlotCoordinates()

```
double VectSharp.Plots.LinearCoordinateSystem1D.ToPlotCoordinates (
            double dataPoint )
```

Definition at line 979 of file CoordinateSystems.cs.

7.93.4 Property Documentation

7.93.4.1 Max

```
double VectSharp.Plots.LinearCoordinateSystem1D.Max  [get], [set]
```

The maximum value.

Definition at line 934 of file CoordinateSystems.cs.

7.93.4.2 Min

```
double VectSharp.Plots.LinearCoordinateSystem1D.Min  [get], [set]
```

The minimum value.

Definition at line 929 of file CoordinateSystems.cs.

7.93.4.3 Scale

```
double VectSharp.Plots.LinearCoordinateSystem1D.Scale  [get], [set]
```

The scale factor.

Definition at line 939 of file CoordinateSystems.cs.

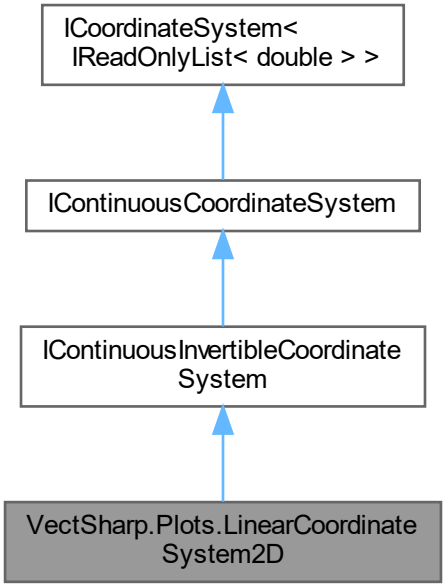
The documentation for this class was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.94 VectSharp.Plots.LinearCoordinateSystem2D Class Reference

Represents a linear coordinate system.

Inheritance diagram for VectSharp.Plots.LinearCoordinateSystem2D:



Public Member Functions

- bool IsDirectionStraight (IReadOnlyList< double > direction)

  Determines whether points aligned along direction in data space are also aligned along some line in plot space.

Parameters

direction	*The direction in data space along which the points should be aligned.*

Returns

`true` if any two points aligned along direction in data space are also aligned along some line in plot space, `false` otherwise.

- LinearCoordinateSystem2D (double minX, double maxX, double minY, double maxY, double scaleX, double scaleY)

  Creates a new LinearCoordinateSystem2D manually specifying the parameter values.
- LinearCoordinateSystem2D (IReadOnlyList< IReadOnlyList< double > > data, double scaleX=350, double scaleY=250)

  Creates a new LinearCoordinateSystem2D determining the value range from the data .
- LinearCoordinateSystem2D (double[,] data, double scaleX=350, double scaleY=250)

  Creates a new LinearCoordinateSystem2D determining the value range from the data .
- double[ ] ToDataCoordinates (Point plotPoint)

  Transform a point in plot space back into data space.

Parameters

plotPoint	*The point in plot space.*

Returns

The point in data space corresponding to the specified point in plot space.

- Point ToPlotCoordinates (IReadOnlyList< double > dataPoint)
- Point ToPlotCoordinates (Point dataPoint)

  Transforms the specified dataPoint into a plot point.
- double[ ] GetAround (IReadOnlyList< double > point, IReadOnlyList< double > direction)

  Gets a data element that is arbitrarily close to the specified point , along the specified direction .

Parameters

point	*The point close to which the returned data element should be.*
direction	*The direction (in data space) along which the returned point should be.*

Returns

A data element that is arbitrarily close to the specified point , along the specified direction .

Properties

- double MinX `[get, set]`

  The minimum X value.
- double MaxX `[get, set]`

  The maximum X value.
- double MinY `[get, set]`

  The minimum Y value.
- double MaxY `[get, set]`

  The maximum Y value.
- double ScaleX `[get, set]`

  The X scale.
- double ScaleY `[get, set]`

  The y scale.
- bool IsLinear `[get]`

  Gets whether the current coordinate system is linear along all directions.
- double[ ] Resolution `[get, set]`

  The maximum difference between two points in data space that appear arbitrarily close in plot space, or some approximation.

7.94.1 Detailed Description

Represents a linear coordinate system.

Definition at line 134 of file CoordinateSystems.cs.

7.94.2 Constructor & Destructor Documentation

7.94.2.1 LinearCoordinateSystem2D() [1/3]

```
VectSharp.Plots.LinearCoordinateSystem2D.LinearCoordinateSystem2D (
            double minX,
            double maxX,
            double minY,
            double maxY,
            double scaleX,
            double scaleY )
```

Creates a new LinearCoordinateSystem2D manually specifying the parameter values.

Parameters

minX	The minimum X value.
maxX	The maximum X value.
minY	The minimum Y value.
maxY	The maximum Y value.
scaleX	The X scale.
scaleY	The Y scale.

Definition at line 197 of file CoordinateSystems.cs.

7.94.2.2 LinearCoordinateSystem2D() [2/3]

```
VectSharp.Plots.LinearCoordinateSystem2D.LinearCoordinateSystem2D (
            IReadOnlyList< IReadOnlyList< double > > data,
            double scaleX = 350,
            double scaleY = 250 )
```

Creates a new LinearCoordinateSystem2D determining the value range from the *data* .

Parameters

data	The data from which the value range should be determined.
scaleX	The X scale.
scaleY	The Y scale.

Definition at line 213 of file CoordinateSystems.cs.

7.94.2.3 LinearCoordinateSystem2D() [3/3]

```
VectSharp.Plots.LinearCoordinateSystem2D.LinearCoordinateSystem2D (
            double data[,],
            double scaleX = 350,
            double scaleY = 250 )
```

Creates a new LinearCoordinateSystem2D determining the value range from the *data* .

Parameters

data	The data from which the value range should be determined.
scaleX	The X scale.
scaleY	The Y scale.

Definition at line 249 of file CoordinateSystems.cs.

7.94.3 Member Function Documentation

7.94.3.1 GetAround()

```
double[] VectSharp.Plots.LinearCoordinateSystem2D.GetAround (
            IReadOnlyList< double > point,
            IReadOnlyList< double > direction )
```

Gets a data element that is arbitrarily close to the specified *point* , along the specified *direction* .

Parameters

point	The point close to which the returned data element should be.
direction	The direction (in data space) along which the returned point should be.

Returns

A data element that is arbitrarily close to the specified *point* , along the specified *direction* .

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 302 of file CoordinateSystems.cs.

7.94.3.2 IsDirectionStraight()

```
bool VectSharp.Plots.LinearCoordinateSystem2D.IsDirectionStraight (
            IReadOnlyList< double > direction )
```

Determines whether points aligned along *direction* in data space are also aligned along some line in plot space.

Parameters

direction	The direction in data space along which the points should be aligned.

Returns

`true` if any two points aligned along *direction* in data space are also aligned along some line in plot space, `false` otherwise.

Implements VectSharp.Plots.IContinuousCoordinateSystem.

7.94.3.3 ToDataCoordinates()

```
double[] VectSharp.Plots.LinearCoordinateSystem2D.ToDataCoordinates (
            Point plotPoint )
```

Transform a point in plot space back into data space.

Parameters

plotPoint	The point in plot space.

Returns

The point in data space corresponding to the specified point in plot space.

Implements VectSharp.Plots.IContinuousInvertibleCoordinateSystem.

Definition at line 280 of file CoordinateSystems.cs.

7.94.3.4 ToPlotCoordinates() [1/2]

```
Point VectSharp.Plots.LinearCoordinateSystem2D.ToPlotCoordinates (
            IReadOnlyList< double > dataPoint )
```

Definition at line 286 of file CoordinateSystems.cs.

7.94.3.5 ToPlotCoordinates() [2/2]

```
Point VectSharp.Plots.LinearCoordinateSystem2D.ToPlotCoordinates (
            Point dataPoint )
```

Transforms the specified *dataPoint* into a plot point.

Parameters

dataPoint	The data whose plot coordinates should be determined.

Returns

A Point representing the *dataPoint* in plot space.

Definition at line 296 of file CoordinateSystems.cs.

7.94.4 Property Documentation

7.94.4.1 IsLinear

```
bool VectSharp.Plots.LinearCoordinateSystem2D.IsLinear  [get]
```

Gets whether the current coordinate system is linear along all directions.

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 167 of file CoordinateSystems.cs.

7.94.4.2 MaxX

```
double VectSharp.Plots.LinearCoordinateSystem2D.MaxX  [get], [set]
```

The maximum X value.

Definition at line 144 of file CoordinateSystems.cs.

7.94.4.3 MaxY

```
double VectSharp.Plots.LinearCoordinateSystem2D.MaxY  [get], [set]
```

The maximum Y value.

Definition at line 154 of file CoordinateSystems.cs.

7.94.4.4 MinX

```
double VectSharp.Plots.LinearCoordinateSystem2D.MinX [get], [set]
```

The minimum X value.

Definition at line 139 of file CoordinateSystems.cs.

7.94.4.5 MinY

```
double VectSharp.Plots.LinearCoordinateSystem2D.MinY [get], [set]
```

The minimum Y value.

Definition at line 149 of file CoordinateSystems.cs.

7.94.4.6 Resolution

```
double [] VectSharp.Plots.LinearCoordinateSystem2D.Resolution [get], [set]
```

The maximum difference between two points in data space that appear arbitrarily close in plot space, or some approximation.

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 175 of file CoordinateSystems.cs.

7.94.4.7 ScaleX

```
double VectSharp.Plots.LinearCoordinateSystem2D.ScaleX [get], [set]
```

The X scale.

Definition at line 159 of file CoordinateSystems.cs.

7.94.4.8 ScaleY

```
double VectSharp.Plots.LinearCoordinateSystem2D.ScaleY [get], [set]
```

The y scale.

Definition at line 164 of file CoordinateSystems.cs.

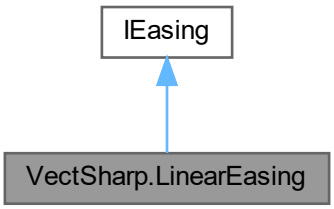
The documentation for this class was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.95 VectSharp.LinearEasing Class Reference

Describes a linear easing (i.e., no easing).

Inheritance diagram for VectSharp.LinearEasing:



Public Member Functions

- LinearEasing ()

  Creates a new LinearEasing.
- double Ease (double value)

  Applies the easing to the specified transition offset.

Parameters

value	*The transition offset (ranging from 0 to 1).*

Returns

The eased transition offset value.

7.95.1 Detailed Description

Describes a linear easing (i.e., no easing).

Definition at line 1353 of file Animation.cs.

7.95.2 Constructor & Destructor Documentation

7.95.2.1 LinearEasing()

```
VectSharp.LinearEasing.LinearEasing ( )
```

Creates a new LinearEasing.

Definition at line 1358 of file Animation.cs.

7.95.3 Member Function Documentation

7.95.3.1 Ease()

```
double VectSharp.LinearEasing.Ease (
            double value )
```

Applies the easing to the specified transition offset.

Parameters

value	The transition offset (ranging from 0 to 1).

Returns

The eased transition offset value.

Implements VectSharp.IEasing.

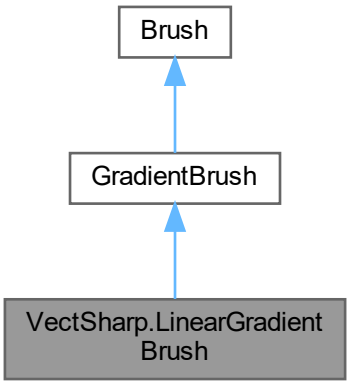
The documentation for this class was generated from the following file:

- VectSharp/Animation.cs

7.96 VectSharp.LinearGradientBrush Class Reference

Represents a brush painting with a linear gradient.

Inheritance diagram for VectSharp.LinearGradientBrush:



Public Member Functions

- LinearGradientBrush (Point startPoint, Point endPoint, IEnumerable< GradientStop > gradientStops)

  Creates a new LinearGradientBrush with the specified start point, end point and gradient stops.
- LinearGradientBrush (Point startPoint, Point endPoint, params GradientStop[ ] gradientStops)

  Creates a new LinearGradientBrush with the specified start point, end point and gradient stops.
- LinearGradientBrush RelativeTo (Graphics referenceGraphics)

  Returns a LinearGradientBrush with the same gradient stops as the current instance, whose start and end point correspond to the points of the current instance in the original reference frame of the referenceGraphics . This involves computing the current transform matrix of the referenceGraphics , inverting it, and applying the inverse matrix to the StartPoint and EndPoint of the current instance.
- override Brush MultiplyOpacity (double opacity)

  Returns a brush corresponding the current instance, with the specified opacity multiplication applied.

Parameters

opacity	*The value that will be used to multiply the opacity of the brush.*

Returns

A brush corresponding the current instance, with the specified opacity multiplication applied.

Properties

- Point StartPoint `[get]`

  The starting point of the gradient. Note that this is relative to the current coordinate system when the gradient is used.
- Point EndPoint `[get]`

  The end point of the gradient. Note that this is relative to the current coordinate system when the gradient is used.

Additional Inherited Members

7.96.1 Detailed Description

Represents a brush painting with a linear gradient.

Definition at line 291 of file Brush.cs.

7.96.2 Constructor & Destructor Documentation

7.96.2.1 LinearGradientBrush() [1/2]

```
VectSharp.LinearGradientBrush.LinearGradientBrush (
            Point startPoint,
            Point endPoint,
            IEnumerable< GradientStop > gradientStops )
```

Creates a new LinearGradientBrush with the specified start point, end point and gradient stops.

Parameters

startPoint	The starting point of the gradient. Note that this is relative to the current coordinate system when the gradient is used.
endPoint	The ending point of the gradient. Note that this is relative to the current coordinate system when the gradient is used.
gradientStops	The colour stops in the gradient.

Definition at line 309 of file Brush.cs.

7.96.2.2 LinearGradientBrush() [2/2]

```
VectSharp.LinearGradientBrush.LinearGradientBrush (
            Point startPoint,
            Point endPoint,
            params GradientStop[] gradientStops )
```

Creates a new LinearGradientBrush with the specified start point, end point and gradient stops.

Parameters

startPoint	The starting point of the gradient. Note that this is relative to the current coordinate system when the gradient is used.
endPoint	The ending point of the gradient. Note that this is relative to the current coordinate system when the gradient is used.
gradientStops	The colour stops in the gradient.

Definition at line 323 of file Brush.cs.

7.96.3 Member Function Documentation

7.96.3.1 MultiplyOpacity()

```
override Brush VectSharp.LinearGradientBrush.MultiplyOpacity (
            double opacity )  [virtual]
```

Returns a brush corresponding the current instance, with the specified *opacity* multiplication applied.

Parameters

opacity	The value that will be used to multiply the opacity of the brush.

Returns

A brush corresponding the current instance, with the specified *opacity* multiplication applied.

Implements VectSharp.Brush.

Definition at line 406 of file Brush.cs.

7.96.3.2 RelativeTo()

```
LinearGradientBrush VectSharp.LinearGradientBrush.RelativeTo (
            Graphics referenceGraphics )
```

Returns a LinearGradientBrush with the same gradient stops as the current instance, whose start and end point correspond to the points of the current instance in the original reference frame of the *referenceGraphics* . This involves computing the current transform matrix of the *referenceGraphics* , inverting it, and applying the inverse matrix to the StartPoint and EndPoint of the current instance.

Parameters

referenceGraphics	The Graphics whose original reference frame is to be used.

Returns

A LinearGradientBrush with the same gradient stops as the current instance, whose start and end point correspond to the points of the current instance in the original reference frame of the *referenceGraphics* .

Definition at line 356 of file Brush.cs.

7.96.4 Property Documentation

7.96.4.1 EndPoint

```
Point VectSharp.LinearGradientBrush.EndPoint  [get]
```

The end point of the gradient. Note that this is relative to the current coordinate system when the gradient is used.

Definition at line 301 of file Brush.cs.

7.96.4.2 StartPoint

```
Point VectSharp.LinearGradientBrush.StartPoint  [get]
```

The starting point of the gradient. Note that this is relative to the current coordinate system when the gradient is used.

Definition at line 296 of file Brush.cs.

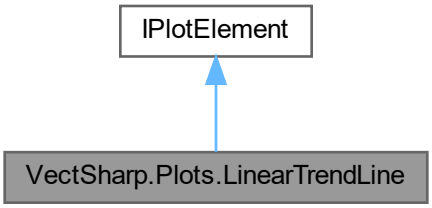
The documentation for this class was generated from the following file:

- VectSharp/Brush.cs

7.97 VectSharp.Plots.LinearTrendLine Class Reference

A plot element that draws a linear trendline with equation `y = a * x + b`.

Inheritance diagram for VectSharp.Plots.LinearTrendLine:



Public Member Functions

- LinearTrendLine (double slope, double intercept, double minX, double minY, double maxX, double maxY, IContinuousCoordinateSystem coordinateSystem)

  Create a new LinearTrendLine instance, specifying the equation parameters.
- LinearTrendLine (IReadOnlyList< IReadOnlyList< double > > data, IContinuousCoordinateSystem coordinateSystem, double? fixedIntercept=null)

  Create a new LinearTrendLine instance, determining the equation parameters by running a regression.
- LinearTrendLine (IReadOnlyList<(double, double)> data, IContinuousCoordinateSystem coordinateSystem, double? fixedIntercept=null)

  Create a new LinearTrendLine instance, determining the equation parameters by running a regression.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- double Slope [get, set]

  The slope of the trendline (a).
- double Intercept [get, set]

  The intercept of the trendline (b).
- double MinX [get, set]

  The minimum X value for which the trendline is plotted.
- double MinY [get, set]

  The minimum Y value for which the trendline is plotted.
- double MaxX [get, set]

  The maximum X value for which the trendline is plotted.
- double MaxY [get, set]

  The maximum Y value for which the trendline is plotted.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180, 180, 180) } [get, set]

  Presentation attributes for the trendline.
- string Tag [get, set]

  A tag to identify the trendline in the plot.
- IContinuousCoordinateSystem CoordinateSystem [get, set]

  The coordinate system used to transform the points from data space to plot space.

7.97.1 Detailed Description

A plot element that draws a linear trendline with equation `y = a * x + b`.

Definition at line 31 of file Trendlines.cs.

7.97.2 Constructor & Destructor Documentation

7.97.2.1 LinearTrendLine() [1/3]

```
VectSharp.Plots.LinearTrendLine.LinearTrendLine (
            double slope,
            double intercept,
            double minX,
            double minY,
            double maxX,
            double maxY,
            IContinuousCoordinateSystem coordinateSystem )
```

Create a new LinearTrendLine instance, specifying the equation parameters.

Parameters

slope	The slope of the trendline (a).
intercept	The intercept of the trendline (b).
minX	The minimum X value for which the trendline is plotted.
minY	The minimum Y value for which the trendline is plotted.
maxX	The maximum X value for which the trendline is plotted.
maxY	The maximum Y value for which the trendline is plotted.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 89 of file Trendlines.cs.

7.97.2.2 LinearTrendLine() [2/3]

```
VectSharp.Plots.LinearTrendLine.LinearTrendLine (
            IReadOnlyList< IReadOnlyList< double > > data,
            IContinuousCoordinateSystem coordinateSystem,
            double?  fixedIntercept = null )
```

Create a new LinearTrendLine instance, determining the equation parameters by running a regression.

Parameters

data	The data that will be used to determine the equation parameters.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.
fixedIntercept	If this is `null`, the intercept (b) is determined during the regression; otherwise, it is fixed to the specified value.

Definition at line 106 of file Trendlines.cs.

7.97.2.3 LinearTrendLine() [3/3]

```
VectSharp.Plots.LinearTrendLine.LinearTrendLine (
            IReadOnlyList<(double, double)> data,
            IContinuousCoordinateSystem coordinateSystem,
            double?  fixedIntercept = null )
```

Create a new LinearTrendLine instance, determining the equation parameters by running a regression.

Parameters

data	The data that will be used to determine the equation parameters.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.
fixedIntercept	If this is `null`, the intercept (b) is determined during the regression; otherwise, it is fixed to the specified value.

Definition at line 154 of file Trendlines.cs.

7.97.3 Member Function Documentation

7.97.3.1 Plot()

```
void VectSharp.Plots.LinearTrendLine.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 157 of file Trendlines.cs.

7.97.4 Property Documentation

7.97.4.1 CoordinateSystem

```
IContinuousCoordinateSystem VectSharp.Plots.LinearTrendLine.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 76 of file Trendlines.cs.

7.97.4.2 Intercept

```
double VectSharp.Plots.LinearTrendLine.Intercept  [get], [set]
```

The intercept of the trendline (b).

Definition at line 41 of file Trendlines.cs.

7.97.4.3 MaxX

```
double VectSharp.Plots.LinearTrendLine.MaxX  [get], [set]
```

The maximum X value for which the trendline is plotted.

Definition at line 56 of file Trendlines.cs.

7.97.4.4 MaxY

```
double VectSharp.Plots.LinearTrendLine.MaxY  [get], [set]
```

The maximum Y value for which the trendline is plotted.

Definition at line 61 of file Trendlines.cs.

7.97.4.5 MinX

```
double VectSharp.Plots.LinearTrendLine.MinX  [get], [set]
```

The minimum X value for which the trendline is plotted.

Definition at line 46 of file Trendlines.cs.

7.97.4.6 MinY

```
double VectSharp.Plots.LinearTrendLine.MinY  [get], [set]
```

The minimum Y value for which the trendline is plotted.

Definition at line 51 of file Trendlines.cs.

7.97.4.7 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.LinearTrendLine.PresentationAttributes = new
PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180,
180, 180) }  [get], [set]
```

Presentation attributes for the trendline.

Definition at line 66 of file Trendlines.cs.

7.97.4.8 Slope

```
double VectSharp.Plots.LinearTrendLine.Slope  [get], [set]
```

The slope of the trendline (a).

Definition at line 36 of file Trendlines.cs.

7.97.4.9 Tag

```
string VectSharp.Plots.LinearTrendLine.Tag  [get], [set]
```

A tag to identify the trendline in the plot.

Definition at line 71 of file Trendlines.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/Trendlines.cs

7.98 VectSharp.LineDash Struct Reference

Represents instructions on how to paint a dashed line.

Public Member Functions

- LineDash (double unitsOn, double unitsOff, double phase)
  Define a new line dash pattern.
- LineDash (double units, double phase=0)
  Define a new line dash pattern.
- LineDash (double[ ] dashArray, double phase=0)
  Define a new line dash pattern.

Public Attributes

- double UnitsOn
  Length of the "on" (painted) segment.
- double UnitsOff
  Length of the "off" (not painted) segment.
- double[ ] DashArray
  An array specifying lenghts of alternating dashes and gaps.
- double Phase
  Position in the dash pattern at which the line starts.

Static Public Attributes

- static LineDash SolidLine = new LineDash(0, 0, 0)
  A solid (not dashed) line

7.98.1 Detailed Description

Represents instructions on how to paint a dashed line.

Definition at line 232 of file Enums.cs.

7.98.2 Constructor & Destructor Documentation

7.98.2.1 LineDash() [1/3]

```
VectSharp.LineDash.LineDash (
            double unitsOn,
            double unitsOff,
            double phase )
```

Define a new line dash pattern.

Parameters

unitsOn	The length of the "on" (painted) segment.
unitsOff	The length of the "off" (not painted) segment.
phase	The position in the dash pattern at which the line starts.

Definition at line 267 of file Enums.cs.

7.98.2.2 LineDash() [2/3]

```
VectSharp.LineDash.LineDash (
            double units,
            double phase = 0 )
```

Define a new line dash pattern.

Parameters

units	The length of the dash segments and gaps.
phase	The position in the dash pattern at which the line starts.

Definition at line 284 of file Enums.cs.

7.98.2.3 LineDash() [3/3]

```
VectSharp.LineDash.LineDash (
            double[] dashArray,
            double phase = 0 )
```

Define a new line dash pattern.

Parameters

dashArray	An array specifying lenghts of alternating dashes and gaps.
phase	The position in the dash pattern at which the line starts.

Definition at line 301 of file Enums.cs.

7.98.3 Member Data Documentation

7.98.3.1 DashArray

```
double [] VectSharp.LineDash.DashArray
```

An array specifying lenghts of alternating dashes and gaps.

Definition at line 254 of file Enums.cs.

7.98.3.2 Phase

```
double VectSharp.LineDash.Phase
```

Position in the dash pattern at which the line starts.

Definition at line 259 of file Enums.cs.

7.98.3.3 SolidLine

```
LineDash VectSharp.LineDash.SolidLine = new LineDash(0, 0, 0)  [static]
```

A solid (not dashed) line

Definition at line 237 of file Enums.cs.

7.98.3.4 UnitsOff

```
double VectSharp.LineDash.UnitsOff
```

Length of the "off" (not painted) segment.

Definition at line 249 of file Enums.cs.

7.98.3.5 UnitsOn

```
double VectSharp.LineDash.UnitsOn
```

Length of the "on" (painted) segment.

Definition at line 243 of file Enums.cs.

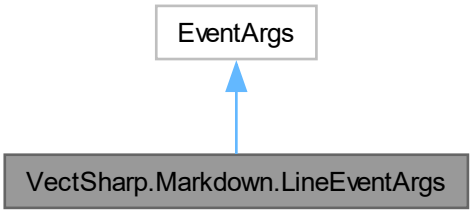
The documentation for this struct was generated from the following file:

- VectSharp/Enums.cs

7.99 VectSharp.Markdown.LineEventArgs Class Reference

EventArgs for the MarkdownRenderer.LineRendering and MarkdownRenderer.LineRendered events.

Inheritance diagram for VectSharp.Markdown.LineEventArgs:



Public Member Functions

- LineEventArgs (MarkdownContext markdownContext, Graphics graphics, Line line, double pageMaxX, double pageMaxY, double contentMaxX, double contentMinY, double contentMaxY, double baseline)

  Create a new LineEventArgs

Properties

- MarkdownContext MarkdownContext `[get, set]`

  The current VectSharp.Markdown.MarkdownContext.
- Graphics Graphics `[get, set]`

  The current VectSharp.Graphics surface.
- Line Line `[get]`

  The line that will be rendered.
- double PageMaxX `[get]`

  The maximum space available for the line on the x axis.
- double PageMaxY `[get]`

The maximum space available for the line on the y axis.

- double ContentMaxX [get]

  The maximum x coordinate of the content in the line.

- double ContentMinY [get]

  The minimum y coordinate of the content in the line.

- double ContentMaxY [get]

  The maximum y coordinate of the content in the line.

- double Baseline [get]

  The vertical coordinate of the baseline.

7.99.1 Detailed Description

EventArgs for the MarkdownRenderer.LineRendering and MarkdownRenderer.LineRendered events.

Definition at line 517 of file MarkdownRenderer.Events.cs.

7.99.2 Constructor & Destructor Documentation

7.99.2.1 LineEventArgs()

```
VectSharp.Markdown.LineEventArgs.LineEventArgs (
            MarkdownContext markdownContext,
            Graphics graphics,
            Line line,
            double pageMaxX,
            double pageMaxY,
            double contentMaxX,
            double contentMinY,
            double contentMaxY,
            double baseline )
```

Create a new LineEventArgs

Parameters

markdownContext	The current VectSharp.Markdown.MarkdownContext.
graphics	The current VectSharp.Graphics surface.
line	The line that will be rendered.
pageMaxX	The maximum space available for the line on the x axis.
pageMaxY	The maximum space available for the line on the y axis.
contentMaxX	The maximum x coordinate of the content in the line.
contentMinY	The minimum y coordinate of the content in the line.
contentMaxY	The maximum y coordinate of the content in the line.
baseline	The vertical coordinate of the baseline.

Definition at line 576 of file MarkdownRenderer.Events.cs.

7.99.3 Property Documentation

7.99.3.1 Baseline

```
double VectSharp.Markdown.LineEventArgs.Baseline  [get]
```

The vertical coordinate of the baseline.

Definition at line 562 of file MarkdownRenderer.Events.cs.

7.99.3.2 ContentMaxX

```
double VectSharp.Markdown.LineEventArgs.ContentMaxX  [get]
```

The maximum x coordinate of the content in the line.

Definition at line 547 of file MarkdownRenderer.Events.cs.

7.99.3.3 ContentMaxY

```
double VectSharp.Markdown.LineEventArgs.ContentMaxY  [get]
```

The maximum y coordinate of the content in the line.

Definition at line 557 of file MarkdownRenderer.Events.cs.

7.99.3.4 ContentMinY

```
double VectSharp.Markdown.LineEventArgs.ContentMinY  [get]
```

The minimum y coordinate of the content in the line.

Definition at line 552 of file MarkdownRenderer.Events.cs.

7.99.3.5 Graphics

```
Graphics VectSharp.Markdown.LineEventArgs.Graphics  [get], [set]
```

The current VectSharp.Graphics surface.

Definition at line 527 of file MarkdownRenderer.Events.cs.

7.99.3.6 Line

```
Line VectSharp.Markdown.LineEventArgs.Line  [get]
```

The line that will be rendered.

Definition at line 532 of file MarkdownRenderer.Events.cs.

7.99.3.7 MarkdownContext

```
MarkdownContext VectSharp.Markdown.LineEventArgs.MarkdownContext  [get], [set]
```

The current VectSharp.Markdown.MarkdownContext.

Definition at line 522 of file MarkdownRenderer.Events.cs.

7.99.3.8 PageMaxX

```
double VectSharp.Markdown.LineEventArgs.PageMaxX  [get]
```

The maximum space available for the line on the x axis.

Definition at line 537 of file MarkdownRenderer.Events.cs.

7.99.3.9 PageMaxY

```
double VectSharp.Markdown.LineEventArgs.PageMaxY  [get]
```

The maximum space available for the line on the y axis.

Definition at line 542 of file MarkdownRenderer.Events.cs.

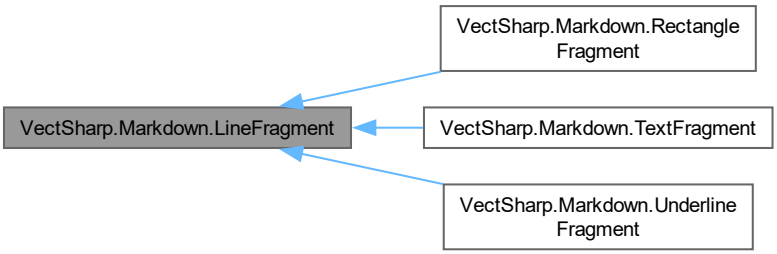
The documentation for this class was generated from the following file:

- VectSharp.Markdown/MarkdownRenderer.Events.cs

7.100 VectSharp.Markdown.LineFragment Class Reference

Represents one of the fragments that make up a Line.

Inheritance diagram for VectSharp.Markdown.LineFragment:



Public Member Functions

- abstract void Translate (double deltaX, double deltaY)

  Applies the specified translation to the line fragment.
- abstract double GetAscent (double lineAscent)

  Get the ascent of the LineFragment.
- abstract double GetMaxY ()

  Get the vertical coordinate of the bottom of the LineFragment.
- abstract double GetMaxX ()

  Get the horizontal coordinate of the right side of the LineFragment.
- abstract ? Point Render (Graphics graphics, double deltaY)

  Render the LineFragment on the specified graphics surface.

Properties

- string Tag `[get]`

  Graphics action tag for the fragment.

7.100.1 Detailed Description

Represents one of the fragments that make up a Line.

Definition at line 26 of file Line.cs.

7.100.2 Member Function Documentation

7.100.2.1 GetAscent()

```
abstract double VectSharp.Markdown.LineFragment.GetAscent (
            double lineAscent )  [pure virtual]
```

Get the ascent of the LineFragment.

Parameters

lineAscent	The ascent of the Line on which the fragment should be drawn.

Returns

The ascent of the LineFragment.

Implemented in VectSharp.Markdown.TextFragment, VectSharp.Markdown.UnderlineFragment, and VectSharp.Markdown.RectangleF

7.100.2.2 GetMaxX()

```
abstract double VectSharp.Markdown.LineFragment.GetMaxX ( )  [pure virtual]
```

Get the horizontal coordinate of the right side of the LineFragment.

Returns

The horizontal coordinate of the right side of the LineFragment.

Implemented in VectSharp.Markdown.TextFragment, VectSharp.Markdown.UnderlineFragment, and VectSharp.Markdown.RectangleF

7.100.2.3 GetMaxY()

```
abstract double VectSharp.Markdown.LineFragment.GetMaxY ( )  [pure virtual]
```

Get the vertical coordinate of the bottom of the LineFragment.

Returns

The vertical coordinates of the bottom of the LineFragment.

Implemented in VectSharp.Markdown.TextFragment, VectSharp.Markdown.UnderlineFragment, and VectSharp.Markdown.RectangleF

7.100.2.4 Render()

```
abstract ?  Point VectSharp.Markdown.LineFragment.Render (
             Graphics graphics,
             double deltaY )  [pure virtual]
```

Render the LineFragment on the specified *graphics* surface.

Parameters

graphics	The Graphics surface on which the LineFragment will be drawn.
deltaY	Additional translation on the vertical axis.

Returns

If a text element has been drawn, this method should return the bottom-right corner of the drawn text element, otherwise `null`. This is used to update the MarkdownContext.BottomRight property.

Implemented in VectSharp.Markdown.TextFragment, VectSharp.Markdown.UnderlineFragment, and VectSharp.Markdown.RectangleF

7.100.2.5 Translate()

```
abstract void VectSharp.Markdown.LineFragment.Translate (
            double deltaX,
            double deltaY )  [pure virtual]
```

Applies the specified translation to the line fragment.

Parameters

deltaX	The translation in the horizontal direction.
deltaY	The translation in the vertical direction.

Implemented in VectSharp.Markdown.TextFragment, VectSharp.Markdown.UnderlineFragment, and VectSharp.Markdown.RectangleF

7.100.3 Property Documentation

7.100.3.1 Tag

```
string VectSharp.Markdown.LineFragment.Tag  [get]
```

Graphics action tag for the fragment.

Definition at line 38 of file Line.cs.

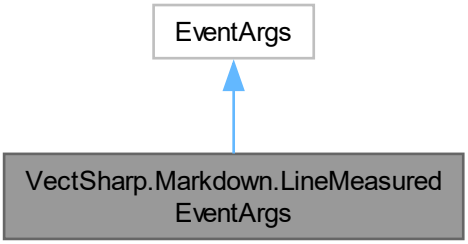
The documentation for this class was generated from the following file:

- VectSharp.Markdown/Line.cs

7.101 VectSharp.Markdown.LineMeasuredEventArgs Class Reference

EventArgs for the MarkdownRenderer.LineMeasured event.

Inheritance diagram for VectSharp.Markdown.LineMeasuredEventArgs:



Public Member Functions

- LineMeasuredEventArgs (MarkdownContext markdownContext, Graphics graphics, Line line, double pageMaxX, double pageMaxY, double contentMaxX, double contentMinY, double contentMaxY, double baseline)

  Create a new LineMeasuredEventArgs

Properties

- MarkdownContext MarkdownContext `[get, set]`

  The current VectSharp.Markdown.MarkdownContext.
- Graphics Graphics `[get, set]`

  The current VectSharp.Graphics surface.
- Line Line `[get, set]`

  The line that will be rendered.
- double PageMaxX `[get]`

  The maximum space available for the line on the x axis.
- double PageMaxY `[get]`

  The maximum space available for the line on the y axis.
- double ContentMaxX `[get]`

  The maximum x coordinate of the content in the line.
- double ContentMinY `[get]`

  The minimum y coordinate of the content in the line.
- double ContentMaxY `[get]`

  The maximum y coordinate of the content in the line.
- double Baseline `[get]`

  The vertical coordinate of the baseline.

7.101.1 Detailed Description

EventArgs for the MarkdownRenderer.LineMeasured event.

Definition at line 441 of file MarkdownRenderer.Events.cs.

7.101.2 Constructor & Destructor Documentation

7.101.2.1 LineMeasuredEventArgs()

```
VectSharp.Markdown.LineMeasuredEventArgs.LineMeasuredEventArgs (
            MarkdownContext markdownContext,
            Graphics graphics,
            Line line,
            double pageMaxX,
            double pageMaxY,
            double contentMaxX,
            double contentMinY,
            double contentMaxY,
            double baseline )
```

Create a new LineMeasuredEventArgs

Parameters

markdownContext	The current VectSharp.Markdown.MarkdownContext.
graphics	The current VectSharp.Graphics surface.
line	The line that will be rendered.
pageMaxX	The maximum space available for the line on the x axis.
pageMaxY	The maximum space available for the line on the y axis.
contentMaxX	The maximum x coordinate of the content in the line.
contentMinY	The minimum y coordinate of the content in the line.
contentMaxY	The maximum y coordinate of the content in the line.
baseline	The vertical coordinate of the baseline.

Definition at line 500 of file MarkdownRenderer.Events.cs.

7.101.3 Property Documentation

7.101.3.1 Baseline

```
double VectSharp.Markdown.LineMeasuredEventArgs.Baseline  [get]
```

The vertical coordinate of the baseline.

Definition at line 486 of file MarkdownRenderer.Events.cs.

7.101.3.2 ContentMaxX

```
double VectSharp.Markdown.LineMeasuredEventArgs.ContentMaxX  [get]
```

The maximum x coordinate of the content in the line.

Definition at line 471 of file MarkdownRenderer.Events.cs.

7.101.3.3 ContentMaxY

```
double VectSharp.Markdown.LineMeasuredEventArgs.ContentMaxY  [get]
```

The maximum y coordinate of the content in the line.

Definition at line 481 of file MarkdownRenderer.Events.cs.

7.101.3.4 ContentMinY

```
double VectSharp.Markdown.LineMeasuredEventArgs.ContentMinY  [get]
```

The minimum y coordinate of the content in the line.

Definition at line 476 of file MarkdownRenderer.Events.cs.

7.101.3.5 Graphics

```
Graphics VectSharp.Markdown.LineMeasuredEventArgs.Graphics  [get], [set]
```

The current VectSharp.Graphics surface.

Definition at line 451 of file MarkdownRenderer.Events.cs.

7.101.3.6 Line

```
Line VectSharp.Markdown.LineMeasuredEventArgs.Line  [get], [set]
```

The line that will be rendered.

Definition at line 456 of file MarkdownRenderer.Events.cs.

7.101.3.7 MarkdownContext

```
MarkdownContext VectSharp.Markdown.LineMeasuredEventArgs.MarkdownContext  [get], [set]
```

The current VectSharp.Markdown.MarkdownContext.

Definition at line 446 of file MarkdownRenderer.Events.cs.

7.101.3.8 PageMaxX

```
double VectSharp.Markdown.LineMeasuredEventArgs.PageMaxX  [get]
```

The maximum space available for the line on the x axis.

Definition at line 461 of file MarkdownRenderer.Events.cs.

7.101.3.9 PageMaxY

```
double VectSharp.Markdown.LineMeasuredEventArgs.PageMaxY  [get]
```

The maximum space available for the line on the y axis.

Definition at line 466 of file MarkdownRenderer.Events.cs.

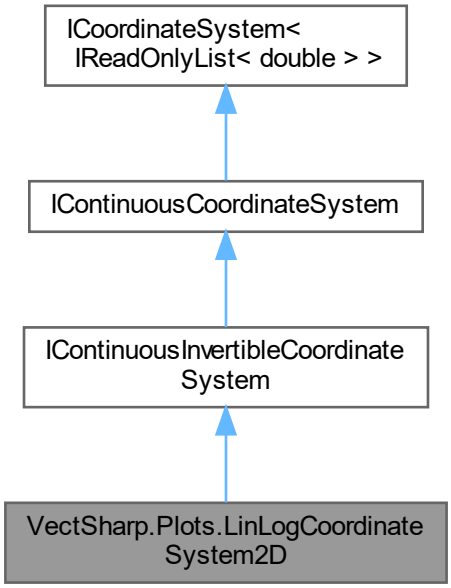
The documentation for this class was generated from the following file:

- VectSharp.Markdown/MarkdownRenderer.Events.cs

7.102 VectSharp.Plots.LinLogCoordinateSystem2D Class Reference

Represents a semi-logarithmic coordinate system with a logarithmic transformation on the X axis.

Inheritance diagram for VectSharp.Plots.LinLogCoordinateSystem2D:



Public Member Functions

- bool IsDirectionStraight (IReadOnlyList< double > direction)

  Determines whether points aligned along direction in data space are also aligned along some line in plot space.

Parameters

direction	*The direction in data space along which the points should be aligned.*

Returns

`true` *if any two points aligned along direction in data space are also aligned along some line in plot space,* `false` *otherwise.*

- LinLogCoordinateSystem2D (double minX, double maxX, double minY, double maxY, double scaleX, double scaleY)

  Creates a new LinLogCoordinateSystem2D manually specifying the parameter values.

- LinLogCoordinateSystem2D (IReadOnlyList< IReadOnlyList< double > > data, double scaleX=350, double scaleY=250)

  Creates a new LinLogCoordinateSystem2D determining the value range from the data .

- LinLogCoordinateSystem2D (double[,] data, double scaleX=350, double scaleY=250)

  Creates a new LinLogCoordinateSystem2D determining the value range from the data .

- double[ ] ToDataCoordinates (Point plotPoint)

  Transform a point in plot space back into data space.

Parameters

plotPoint	*The point in plot space.*

Returns

The point in data space corresponding to the specified point in plot space.

- Point ToPlotCoordinates (IReadOnlyList< double > dataPoint)
- Point ToPlotCoordinates (Point dataPoint)

  Transforms the specified dataPoint into a plot point.
- double[ ] GetAround (IReadOnlyList< double > point, IReadOnlyList< double > direction)

  Gets a data element that is arbitrarily close to the specified point , along the specified direction .

Parameters

point	*The point close to which the returned data element should be.*
direction	*The direction (in data space) along which the returned point should be.*

Returns

A data element that is arbitrarily close to the specified point , along the specified direction .

Properties

- double MinX `[get, set]`

  The minimum X value.
- double MaxX `[get, set]`

  The maximum X value.
- double MinY `[get, set]`

  The minimum Y value (in logarithmic scale).
- double MaxY `[get, set]`

  The maximum Y value (in logarithmic scale).
- double ScaleX `[get, set]`

  The X scale.
- double ScaleY `[get, set]`

  The y scale.
- bool IsLinear `[get]`

  Gets whether the current coordinate system is linear along all directions.
- double[ ] Resolution `[get, set]`

  The maximum difference between two points in data space that appear arbitrarily close in plot space, or some approximation.

7.102.1 Detailed Description

Represents a semi-logarithmic coordinate system with a logarithmic transformation on the X axis.

Definition at line 720 of file CoordinateSystems.cs.

7.102.2 Constructor & Destructor Documentation

7.102.2.1 LinLogCoordinateSystem2D() [1/3]

```
VectSharp.Plots.LinLogCoordinateSystem2D.LinLogCoordinateSystem2D (
            double minX,
            double maxX,
            double minY,
            double maxY,
            double scaleX,
            double scaleY )
```

Creates a new LinLogCoordinateSystem2D manually specifying the parameter values.

Parameters

minX	The minimum X value.
maxX	The maximum X value.
minY	The minimum Y value.
maxY	The maximum Y value.
scaleX	The X scale.
scaleY	The Y scale.

Definition at line 800 of file CoordinateSystems.cs.

7.102.2.2 LinLogCoordinateSystem2D() [2/3]

```
VectSharp.Plots.LinLogCoordinateSystem2D.LinLogCoordinateSystem2D (
            IReadOnlyList< IReadOnlyList< double > > data,
            double scaleX = 350,
            double scaleY = 250 )
```

Creates a new LinLogCoordinateSystem2D determining the value range from the *data* .

Parameters

data	The data from which the value range should be determined.
scaleX	The X scale.
scaleY	The Y scale.

Definition at line 816 of file CoordinateSystems.cs.

7.102.2.3 LinLogCoordinateSystem2D() [3/3]

```
VectSharp.Plots.LinLogCoordinateSystem2D.LinLogCoordinateSystem2D (
            double data[,],
            double scaleX = 350,
            double scaleY = 250 )
```

Creates a new LinLogCoordinateSystem2D determining the value range from the *data* .

Parameters

data	The data from which the value range should be determined.
scaleX	The X scale.
scaleY	The Y scale.

Definition at line 857 of file CoordinateSystems.cs.

7.102.3 Member Function Documentation

7.102.3.1 GetAround()

```
double[] VectSharp.Plots.LinLogCoordinateSystem2D.GetAround (
            IReadOnlyList< double > point,
            IReadOnlyList< double > direction )
```

Gets a data element that is arbitrarily close to the specified *point* , along the specified *direction* .

Parameters

point	The point close to which the returned data element should be.
direction	The direction (in data space) along which the returned point should be.

Returns

A data element that is arbitrarily close to the specified *point* , along the specified *direction* .

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 915 of file CoordinateSystems.cs.

7.102.3.2 IsDirectionStraight()

```
bool VectSharp.Plots.LinLogCoordinateSystem2D.IsDirectionStraight (
            IReadOnlyList< double > direction )
```

Determines whether points aligned along *direction* in data space are also aligned along some line in plot space.

Parameters

direction	The direction in data space along which the points should be aligned.

Returns

`true` if any two points aligned along *direction* in data space are also aligned along some line in plot space, `false` otherwise.

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 756 of file CoordinateSystems.cs.

7.102.3.3 ToDataCoordinates()

```
double[] VectSharp.Plots.LinLogCoordinateSystem2D.ToDataCoordinates (
            Point plotPoint )
```

Transform a point in plot space back into data space.

Parameters

plotPoint	The point in plot space.

Returns

The point in data space corresponding to the specified point in plot space.

Implements VectSharp.Plots.IContinuousInvertibleCoordinateSystem.

Definition at line 893 of file CoordinateSystems.cs.

7.102.3.4 ToPlotCoordinates() [1/2]

```
Point VectSharp.Plots.LinLogCoordinateSystem2D.ToPlotCoordinates (
            IReadOnlyList< double > dataPoint )
```

Definition at line 899 of file CoordinateSystems.cs.

7.102.3.5 ToPlotCoordinates() [2/2]

```
Point VectSharp.Plots.LinLogCoordinateSystem2D.ToPlotCoordinates (
            Point dataPoint )
```

Transforms the specified *dataPoint* into a plot point.

Parameters

dataPoint	The data whose plot coordinates should be determined.

Returns

A Point representing the *dataPoint* in plot space.

Definition at line 909 of file CoordinateSystems.cs.

7.102.4 Property Documentation

7.102.4.1 IsLinear

```
bool VectSharp.Plots.LinLogCoordinateSystem2D.IsLinear  [get]
```

Gets whether the current coordinate system is linear along all directions.

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 753 of file CoordinateSystems.cs.

7.102.4.2 MaxX

```
double VectSharp.Plots.LinLogCoordinateSystem2D.MaxX  [get], [set]
```

The maximum X value.

Definition at line 730 of file CoordinateSystems.cs.

7.102.4.3 MaxY

```
double VectSharp.Plots.LinLogCoordinateSystem2D.MaxY  [get], [set]
```

The maximum Y value (in logarithmic scale).

Definition at line 740 of file CoordinateSystems.cs.

7.102.4.4 MinX

```
double VectSharp.Plots.LinLogCoordinateSystem2D.MinX  [get], [set]
```

The minimum X value.

Definition at line 725 of file CoordinateSystems.cs.

7.102.4.5 MinY

```
double VectSharp.Plots.LinLogCoordinateSystem2D.MinY  [get], [set]
```

The minimum Y value (in logarithmic scale).

Definition at line 735 of file CoordinateSystems.cs.

7.102.4.6 Resolution

```
double [] VectSharp.Plots.LinLogCoordinateSystem2D.Resolution  [get], [set]
```

The maximum difference between two points in data space that appear arbitrarily close in plot space, or some approximation.

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 771 of file CoordinateSystems.cs.

7.102.4.7 ScaleX

```
double VectSharp.Plots.LinLogCoordinateSystem2D.ScaleX  [get], [set]
```

The X scale.

Definition at line 745 of file CoordinateSystems.cs.

7.102.4.8 ScaleY

```
double VectSharp.Plots.LinLogCoordinateSystem2D.ScaleY  [get], [set]
```

The y scale.

Definition at line 750 of file CoordinateSystems.cs.

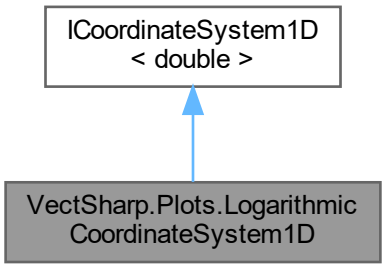
The documentation for this class was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.103 VectSharp.Plots.LogarithmicCoordinateSystem1D Class Reference

Represents a 1-D logarithmic coordinate system.

Inheritance diagram for VectSharp.Plots.LogarithmicCoordinateSystem1D:



Public Member Functions

- LogarithmicCoordinateSystem1D (double min, double max, double scale)

  Creates a new LogarithmicCoordinateSystem1D, manually specifying the parameters.
- LogarithmicCoordinateSystem1D (IReadOnlyList< double > data, double scale=350)

  Creates a new LogarithmicCoordinateSystem1D determining the value range from the data .
- double ToPlotCoordinates (double dataPoint)

Properties

- double Min `[get, set]`

  The minimum value (in logarithmic scale).
- double Max `[get, set]`

  The maximum value (in logarithmic scale).
- double Scale `[get, set]`

  The scale factor.

7.103.1 Detailed Description

Represents a 1-D logarithmic coordinate system.

Definition at line 988 of file CoordinateSystems.cs.

7.103.2 Constructor & Destructor Documentation

7.103.2.1 LogarithmicCoordinateSystem1D() [1/2]

```
VectSharp.Plots.LogarithmicCoordinateSystem1D.LogarithmicCoordinateSystem1D (
            double min,
            double max,
            double scale )
```

Creates a new LogarithmicCoordinateSystem1D, manually specifying the parameters.

Parameters

min	The minimum value.
max	The maximum value.
scale	The scale factor.

Definition at line 1011 of file CoordinateSystems.cs.

7.103.2.2 LogarithmicCoordinateSystem1D() [2/2]

```
VectSharp.Plots.LogarithmicCoordinateSystem1D.LogarithmicCoordinateSystem1D (
            IReadOnlyList< double > data,
            double scale = 350 )
```

Creates a new LogarithmicCoordinateSystem1D determining the value range from the *data* .

Parameters

data	The data from which the value range should be determined.
scale	The scale factor.

Definition at line 1023 of file CoordinateSystems.cs.

7.103.3 Member Function Documentation

7.103.3.1 ToPlotCoordinates()

```
double VectSharp.Plots.LogarithmicCoordinateSystem1D.ToPlotCoordinates (
            double dataPoint )
```

Definition at line 1046 of file CoordinateSystems.cs.

7.103.4 Property Documentation

7.103.4.1 Max

```
double VectSharp.Plots.LogarithmicCoordinateSystem1D.Max  [get], [set]
```

The maximum value (in logarithmic scale).

Definition at line 998 of file CoordinateSystems.cs.

7.103.4.2 Min

```
double VectSharp.Plots.LogarithmicCoordinateSystem1D.Min  [get], [set]
```

The minimum value (in logarithmic scale).

Definition at line 993 of file CoordinateSystems.cs.

7.103.4.3 Scale

```
double VectSharp.Plots.LogarithmicCoordinateSystem1D.Scale  [get], [set]
```

The scale factor.

Definition at line 1003 of file CoordinateSystems.cs.

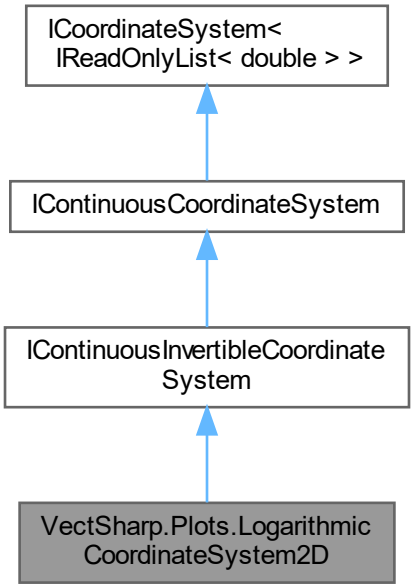
The documentation for this class was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.104 VectSharp.Plots.LogarithmicCoordinateSystem2D Class Reference

Represents a logarithmic coordinate system.

Inheritance diagram for VectSharp.Plots.LogarithmicCoordinateSystem2D:



Public Member Functions

- bool IsDirectionStraight (IReadOnlyList< double > direction)

  Determines whether points aligned along direction in data space are also aligned along some line in plot space.

Parameters

direction	*The direction in data space along which the points should be aligned.*

Returns

`true` *if any two points aligned along direction in data space are also aligned along some line in plot space,* `false` *otherwise.*

- LogarithmicCoordinateSystem2D (double minX, double maxX, double minY, double maxY, double scaleX, double scaleY)

  Creates a new LogarithmicCoordinateSystem2D manually specifying the parameter values.

- LogarithmicCoordinateSystem2D (IReadOnlyList< IReadOnlyList< double > > data, double scaleX=350, double scaleY=250)

  Creates a new LogarithmicCoordinateSystem2D determining the value range from the data .

- LogarithmicCoordinateSystem2D (double[,] data, double scaleX=350, double scaleY=250)

  Creates a new LogarithmicCoordinateSystem2D determining the value range from the data .

- double[ ] ToDataCoordinates (Point plotPoint)

  Transform a point in plot space back into data space.

Parameters

plotPoint	*The point in plot space.*

Returns

The point in data space corresponding to the specified point in plot space.

- Point ToPlotCoordinates (IReadOnlyList< double > dataPoint)
- Point ToPlotCoordinates (Point dataPoint)

  Transforms the specified dataPoint into a plot point.
- double[ ] GetAround (IReadOnlyList< double > point, IReadOnlyList< double > direction)

  Gets a data element that is arbitrarily close to the specified point , along the specified direction .

Parameters

point	*The point close to which the returned data element should be.*
direction	*The direction (in data space) along which the returned point should be.*

Returns

A data element that is arbitrarily close to the specified point , along the specified direction .

Properties

- double MinX `[get, set]`

  The minimum X value (in logarithmic scale).
- double MaxX `[get, set]`

  The maximum X value (in logarithmic scale).
- double MinY `[get, set]`

  The minimum Y value (in logarithmic scale).
- double MaxY `[get, set]`

  The maximum Y value (in logarithmic scale).
- double ScaleX `[get, set]`

  The X scale.
- double ScaleY `[get, set]`

  The y scale.
- bool IsLinear `[get]`

  Gets whether the current coordinate system is linear along all directions.
- double[ ] Resolution `[get, set]`

  The maximum difference between two points in data space that appear arbitrarily close in plot space, or some approximation.

7.104.1 Detailed Description

Represents a logarithmic coordinate system.

Definition at line 311 of file CoordinateSystems.cs.

7.104.2 Constructor & Destructor Documentation

7.104.2.1 LogarithmicCoordinateSystem2D() [1/3]

```
VectSharp.Plots.LogarithmicCoordinateSystem2D.LogarithmicCoordinateSystem2D (
            double minX,
            double maxX,
            double minY,
            double maxY,
            double scaleX,
            double scaleY )
```

Creates a new LogarithmicCoordinateSystem2D manually specifying the parameter values.

Parameters

minX	The minimum X value.
maxX	The maximum X value.
minY	The minimum Y value.
maxY	The maximum Y value.
scaleX	The X scale.
scaleY	The Y scale.

Definition at line 391 of file CoordinateSystems.cs.

7.104.2.2 LogarithmicCoordinateSystem2D() [2/3]

```
VectSharp.Plots.LogarithmicCoordinateSystem2D.LogarithmicCoordinateSystem2D (
            IReadOnlyList< IReadOnlyList< double > > data,
            double scaleX = 350,
            double scaleY = 250 )
```

Creates a new LogarithmicCoordinateSystem2D determining the value range from the *data* .

Parameters

data	The data from which the value range should be determined.
scaleX	The X scale.
scaleY	The Y scale.

Definition at line 407 of file CoordinateSystems.cs.

7.104.2.3 LogarithmicCoordinateSystem2D() [3/3]

```
VectSharp.Plots.LogarithmicCoordinateSystem2D.LogarithmicCoordinateSystem2D (
            double data[,],
            double scaleX = 350,
            double scaleY = 250 )
```

Creates a new LogarithmicCoordinateSystem2D determining the value range from the *data* .

Parameters

data	The data from which the value range should be determined.
scaleX	The X scale.
scaleY	The Y scale.

Definition at line 449 of file CoordinateSystems.cs.

7.104.3 Member Function Documentation

7.104.3.1 GetAround()

```
double[] VectSharp.Plots.LogarithmicCoordinateSystem2D.GetAround (
            IReadOnlyList< double > point,
            IReadOnlyList< double > direction )
```

Gets a data element that is arbitrarily close to the specified *point* , along the specified *direction* .

Parameters

point	The point close to which the returned data element should be.
direction	The direction (in data space) along which the returned point should be.

Returns

A data element that is arbitrarily close to the specified *point* , along the specified *direction* .

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 507 of file CoordinateSystems.cs.

7.104.3.2 IsDirectionStraight()

```
bool VectSharp.Plots.LogarithmicCoordinateSystem2D.IsDirectionStraight (
            IReadOnlyList< double > direction )
```

Determines whether points aligned along *direction* in data space are also aligned along some line in plot space.

Parameters

direction	The direction in data space along which the points should be aligned.

Returns

`true` if any two points aligned along *direction* in data space are also aligned along some line in plot space, `false` otherwise.

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 347 of file CoordinateSystems.cs.

7.104.3.3 ToDataCoordinates()

```
double[] VectSharp.Plots.LogarithmicCoordinateSystem2D.ToDataCoordinates (
            Point plotPoint )
```

Transform a point in plot space back into data space.

Parameters

plotPoint	The point in plot space.

Returns

The point in data space corresponding to the specified point in plot space.

Implements VectSharp.Plots.IContinuousInvertibleCoordinateSystem.

Definition at line 485 of file CoordinateSystems.cs.

7.104.3.4 ToPlotCoordinates() [1/2]

```
Point VectSharp.Plots.LogarithmicCoordinateSystem2D.ToPlotCoordinates (
            IReadOnlyList< double > dataPoint )
```

Definition at line 491 of file CoordinateSystems.cs.

7.104.3.5 ToPlotCoordinates() [2/2]

```
Point VectSharp.Plots.LogarithmicCoordinateSystem2D.ToPlotCoordinates (
            Point dataPoint )
```

Transforms the specified *dataPoint* into a plot point.

Parameters

dataPoint	The data whose plot coordinates should be determined.

Returns

A Point representing the *dataPoint* in plot space.

Definition at line 501 of file CoordinateSystems.cs.

7.104.4 Property Documentation

7.104.4.1 IsLinear

```
bool VectSharp.Plots.LogarithmicCoordinateSystem2D.IsLinear  [get]
```

Gets whether the current coordinate system is linear along all directions.

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 344 of file CoordinateSystems.cs.

7.104.4.2 MaxX

```
double VectSharp.Plots.LogarithmicCoordinateSystem2D.MaxX  [get], [set]
```

The maximum X value (in logarithmic scale).

Definition at line 321 of file CoordinateSystems.cs.

7.104.4.3 MaxY

```
double VectSharp.Plots.LogarithmicCoordinateSystem2D.MaxY  [get], [set]
```

The maximum Y value (in logarithmic scale).

Definition at line 331 of file CoordinateSystems.cs.

7.104.4.4 MinX

```
double VectSharp.Plots.LogarithmicCoordinateSystem2D.MinX [get], [set]
```

The minimum X value (in logarithmic scale).

Definition at line 316 of file CoordinateSystems.cs.

7.104.4.5 MinY

```
double VectSharp.Plots.LogarithmicCoordinateSystem2D.MinY [get], [set]
```

The minimum Y value (in logarithmic scale).

Definition at line 326 of file CoordinateSystems.cs.

7.104.4.6 Resolution

```
double [] VectSharp.Plots.LogarithmicCoordinateSystem2D.Resolution [get], [set]
```

The maximum difference between two points in data space that appear arbitrarily close in plot space, or some approximation.

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 362 of file CoordinateSystems.cs.

7.104.4.7 ScaleX

```
double VectSharp.Plots.LogarithmicCoordinateSystem2D.ScaleX [get], [set]
```

The X scale.

Definition at line 336 of file CoordinateSystems.cs.

7.104.4.8 ScaleY

```
double VectSharp.Plots.LogarithmicCoordinateSystem2D.ScaleY [get], [set]
```

The y scale.

Definition at line 341 of file CoordinateSystems.cs.

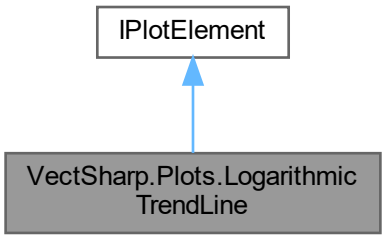
The documentation for this class was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.105 VectSharp.Plots.LogarithmicTrendLine Class Reference

A plot element that draws a logarithmic trendline with equation `y = a * Ln(x) + b`.

Inheritance diagram for VectSharp.Plots.LogarithmicTrendLine:



Public Member Functions

- LogarithmicTrendLine (double slope, double intercept, double minX, double minY, double maxX, double maxY, IContinuousCoordinateSystem coordinateSystem)

  Create a new LogarithmicTrendLine instance, specifying the equation parameters.
- LogarithmicTrendLine (IReadOnlyList< IReadOnlyList< double > > data, IContinuousCoordinateSystem coordinateSystem, double? fixedIntercept=null)

  Create a new LogarithmicTrendLine instance, determining the equation parameters by running a regression.
- LogarithmicTrendLine (IReadOnlyList<(double, double)> data, IContinuousCoordinateSystem coordinate↩ System, double? fixedIntercept=null)

  Create a new LogarithmicTrendLine instance, determining the equation parameters by running a regression.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- double Slope `[get, set]`

  The slope of the trendline (a).
- double Intercept `[get, set]`

  The intercept of the trendline (b).
- double MinX `[get, set]`

  The minimum X value for which the trendline is plotted.
- double MinY `[get, set]`

  The minimum Y value for which the trendline is plotted.
- double MaxX `[get, set]`

The maximum X value for which the trendline is plotted.

- double MaxY `[get, set]`

  The maximum Y value for which the trendline is plotted.

- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180, 180, 180) } `[get, set]`

  Presentation attributes for the trendline.

- string Tag `[get, set]`

  A tag to identify the trendline in the plot.

- IContinuousCoordinateSystem CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.

7.105.1 Detailed Description

A plot element that draws a logarithmic trendline with equation `y = a * Ln(x) + b`.

Definition at line 474 of file Trendlines.cs.

7.105.2 Constructor & Destructor Documentation

7.105.2.1 LogarithmicTrendLine() [1/3]

```
VectSharp.Plots.LogarithmicTrendLine.LogarithmicTrendLine (
            double slope,
            double intercept,
            double minX,
            double minY,
            double maxX,
            double maxY,
            IContinuousCoordinateSystem coordinateSystem )
```

Create a new LogarithmicTrendLine instance, specifying the equation parameters.

Parameters

slope	The slope of the trendline (a).
intercept	The intercept of the trendline (b).
minX	The minimum X value for which the trendline is plotted.
minY	The minimum Y value for which the trendline is plotted.
maxX	The maximum X value for which the trendline is plotted.
maxY	The maximum Y value for which the trendline is plotted.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 532 of file Trendlines.cs.

7.105.2.2 LogarithmicTrendLine() [2/3]

```
VectSharp.Plots.LogarithmicTrendLine.LogarithmicTrendLine (
            IReadOnlyList< IReadOnlyList< double > > data,
            IContinuousCoordinateSystem coordinateSystem,
            double?  fixedIntercept = null )
```

Create a new LogarithmicTrendLine instance, determining the equation parameters by running a regression.

Parameters

data	The data that will be used to determine the equation parameters.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.
fixedIntercept	If this is `null`, the intercept (b) is determined during the regression; otherwise, it is fixed to the specified value.

Definition at line 549 of file Trendlines.cs.

7.105.2.3 LogarithmicTrendLine() [3/3]

```
VectSharp.Plots.LogarithmicTrendLine.LogarithmicTrendLine (
            IReadOnlyList<(double, double)> data,
            IContinuousCoordinateSystem coordinateSystem,
            double?  fixedIntercept = null )
```

Create a new LogarithmicTrendLine instance, determining the equation parameters by running a regression.

Parameters

data	The data that will be used to determine the equation parameters.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.
fixedIntercept	If this is `null`, the intercept (b) is determined during the regression; otherwise, it is fixed to the specified value.

Definition at line 596 of file Trendlines.cs.

7.105.3 Member Function Documentation

7.105.3.1 Plot()

```
void VectSharp.Plots.LogarithmicTrendLine.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 599 of file Trendlines.cs.

7.105.4 Property Documentation

7.105.4.1 CoordinateSystem

```
IContinuousCoordinateSystem VectSharp.Plots.LogarithmicTrendLine.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 519 of file Trendlines.cs.

7.105.4.2 Intercept

```
double VectSharp.Plots.LogarithmicTrendLine.Intercept  [get], [set]
```

The intercept of the trendline (b).

Definition at line 484 of file Trendlines.cs.

7.105.4.3 MaxX

```
double VectSharp.Plots.LogarithmicTrendLine.MaxX  [get], [set]
```

The maximum X value for which the trendline is plotted.

Definition at line 499 of file Trendlines.cs.

7.105.4.4 MaxY

```
double VectSharp.Plots.LogarithmicTrendLine.MaxY  [get], [set]
```

The maximum Y value for which the trendline is plotted.

Definition at line 504 of file Trendlines.cs.

7.105.4.5 MinX

```
double VectSharp.Plots.LogarithmicTrendLine.MinX  [get], [set]
```

The minimum X value for which the trendline is plotted.

Definition at line 489 of file Trendlines.cs.

7.105.4.6 MinY

```
double VectSharp.Plots.LogarithmicTrendLine.MinY  [get], [set]
```

The minimum Y value for which the trendline is plotted.

Definition at line 494 of file Trendlines.cs.

7.105.4.7 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.LogarithmicTrendLine.PresentationAttributes
= new PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180,
180, 180) }  [get], [set]
```

Presentation attributes for the trendline.

Definition at line 509 of file Trendlines.cs.

7.105.4.8 Slope

```
double VectSharp.Plots.LogarithmicTrendLine.Slope  [get], [set]
```

The slope of the trendline (a).

Definition at line 479 of file Trendlines.cs.

7.105.4.9 Tag

```
string VectSharp.Plots.LogarithmicTrendLine.Tag  [get], [set]
```

A tag to identify the trendline in the plot.

Definition at line 514 of file Trendlines.cs.

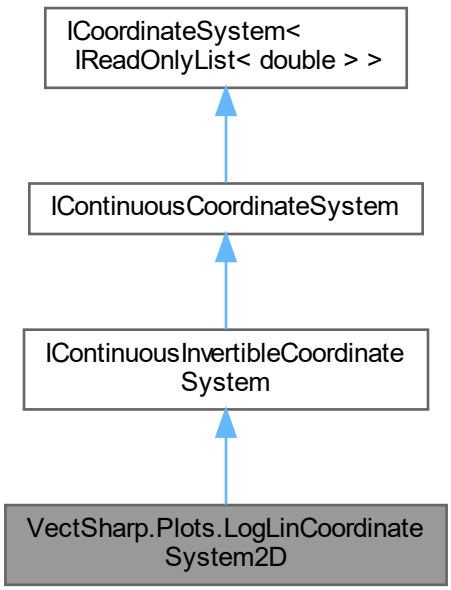
The documentation for this class was generated from the following file:

- VectSharp.Plots/Trendlines.cs

7.106 VectSharp.Plots.LogLinCoordinateSystem2D Class Reference

Represents a semi-logarithmic coordinate system with a logarithmic transformation on the Y axis.

Inheritance diagram for VectSharp.Plots.LogLinCoordinateSystem2D:



Public Member Functions

- bool IsDirectionStraight (IReadOnlyList< double > direction)

  Determines whether points aligned along direction in data space are also aligned along some line in plot space.

Parameters

direction	*The direction in data space along which the points should be aligned.*

Returns

`true` *if any two points aligned along direction in data space are also aligned along some line in plot space,* `false` *otherwise.*

- LogLinCoordinateSystem2D (double minX, double maxX, double minY, double maxY, double scaleX, double scaleY)

  Creates a new LogLinCoordinateSystem2D manually specifying the parameter values.
- LogLinCoordinateSystem2D (IReadOnlyList< IReadOnlyList< double > > data, double scaleX=350, double scaleY=250)

  Creates a new LogLinCoordinateSystem2D determining the value range from the data .
- LogLinCoordinateSystem2D (double[,] data, double scaleX=350, double scaleY=250)

  Creates a new LogLinCoordinateSystem2D determining the value range from the data .
- double[ ] ToDataCoordinates (Point plotPoint)

  Transform a point in plot space back into data space.

Parameters

plotPoint	*The point in plot space.*

Returns

The point in data space corresponding to the specified point in plot space.

- Point ToPlotCoordinates (IReadOnlyList< double > dataPoint)
- Point ToPlotCoordinates (Point dataPoint)

  Transforms the specified dataPoint into a plot point.
- double[ ] GetAround (IReadOnlyList< double > point, IReadOnlyList< double > direction)

  Gets a data element that is arbitrarily close to the specified point , along the specified direction .

Parameters

point	*The point close to which the returned data element should be.*
direction	*The direction (in data space) along which the returned point should be.*

Returns

A data element that is arbitrarily close to the specified point , along the specified direction .

Properties

- double MinX `[get, set]`

  The minimum X value.
- double MaxX `[get, set]`

  The maximum X value.
- double MinY `[get, set]`

  The minimum Y value (in logarithmic scale).
- double MaxY `[get, set]`

  The maximum Y value (in logarithmic scale).
- double ScaleX `[get, set]`

  The X scale.
- double ScaleY `[get, set]`

  The y scale.
- bool IsLinear `[get]`

  Gets whether the current coordinate system is linear along all directions.
- double[ ] Resolution `[get, set]`

  The maximum difference between two points in data space that appear arbitrarily close in plot space, or some approximation.

7.106.1 Detailed Description

Represents a semi-logarithmic coordinate system with a logarithmic transformation on the Y axis.

Definition at line 516 of file CoordinateSystems.cs.

7.106.2 Constructor & Destructor Documentation

7.106.2.1 LogLinCoordinateSystem2D() [1/3]

```
VectSharp.Plots.LogLinCoordinateSystem2D.LogLinCoordinateSystem2D (
            double minX,
            double maxX,
            double minY,
            double maxY,
            double scaleX,
            double scaleY )
```

Creates a new LogLinCoordinateSystem2D manually specifying the parameter values.

Parameters

minX	The minimum X value.
maxX	The maximum X value.
minY	The minimum Y value.
maxY	The maximum Y value.
scaleX	The X scale.
scaleY	The Y scale.

Definition at line 596 of file CoordinateSystems.cs.

7.106.2.2 LogLinCoordinateSystem2D() [2/3]

```
VectSharp.Plots.LogLinCoordinateSystem2D.LogLinCoordinateSystem2D (
            IReadOnlyList< IReadOnlyList< double > > data,
            double scaleX = 350,
            double scaleY = 250 )
```

Creates a new LogLinCoordinateSystem2D determining the value range from the *data* .

Parameters

data	The data from which the value range should be determined.
scaleX	The X scale.
scaleY	The Y scale.

Definition at line 612 of file CoordinateSystems.cs.

7.106.2.3 LogLinCoordinateSystem2D() [3/3]

```
VectSharp.Plots.LogLinCoordinateSystem2D.LogLinCoordinateSystem2D (
            double data[,],
            double scaleX = 350,
            double scaleY = 250 )
```

Creates a new LogLinCoordinateSystem2D determining the value range from the *data* .

Parameters

data	The data from which the value range should be determined.
scaleX	The X scale.
scaleY	The Y scale.

Definition at line 653 of file CoordinateSystems.cs.

7.106.3 Member Function Documentation

7.106.3.1 GetAround()

```
double[] VectSharp.Plots.LogLinCoordinateSystem2D.GetAround (
            IReadOnlyList< double > point,
            IReadOnlyList< double > direction )
```

Gets a data element that is arbitrarily close to the specified *point* , along the specified *direction* .

Parameters

point	The point close to which the returned data element should be.
direction	The direction (in data space) along which the returned point should be.

Returns

A data element that is arbitrarily close to the specified *point* , along the specified *direction* .

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 711 of file CoordinateSystems.cs.

7.106.3.2 IsDirectionStraight()

```
bool VectSharp.Plots.LogLinCoordinateSystem2D.IsDirectionStraight (
            IReadOnlyList< double > direction )
```

Determines whether points aligned along *direction* in data space are also aligned along some line in plot space.

Parameters

direction	The direction in data space along which the points should be aligned.

Returns

`true` if any two points aligned along *direction* in data space are also aligned along some line in plot space, `false` otherwise.

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 552 of file CoordinateSystems.cs.

7.106.3.3 ToDataCoordinates()

```
double[] VectSharp.Plots.LogLinCoordinateSystem2D.ToDataCoordinates (
            Point plotPoint )
```

Transform a point in plot space back into data space.

Parameters

plotPoint	The point in plot space.

Returns

The point in data space corresponding to the specified point in plot space.

Implements VectSharp.Plots.IContinuousInvertibleCoordinateSystem.

Definition at line 689 of file CoordinateSystems.cs.

7.106.3.4 ToPlotCoordinates() [1/2]

```
Point VectSharp.Plots.LogLinCoordinateSystem2D.ToPlotCoordinates (
            IReadOnlyList< double > dataPoint )
```

Definition at line 695 of file CoordinateSystems.cs.

7.106.3.5 ToPlotCoordinates() [2/2]

```
Point VectSharp.Plots.LogLinCoordinateSystem2D.ToPlotCoordinates (
            Point dataPoint )
```

Transforms the specified *dataPoint* into a plot point.

Parameters

dataPoint	The data whose plot coordinates should be determined.

Returns

A Point representing the *dataPoint* in plot space.

Definition at line 705 of file CoordinateSystems.cs.

7.106.4 Property Documentation

7.106.4.1 IsLinear

```
bool VectSharp.Plots.LogLinCoordinateSystem2D.IsLinear  [get]
```

Gets whether the current coordinate system is linear along all directions.

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 549 of file CoordinateSystems.cs.

7.106.4.2 MaxX

```
double VectSharp.Plots.LogLinCoordinateSystem2D.MaxX  [get], [set]
```

The maximum X value.

Definition at line 526 of file CoordinateSystems.cs.

7.106.4.3 MaxY

```
double VectSharp.Plots.LogLinCoordinateSystem2D.MaxY  [get], [set]
```

The maximum Y value (in logarithmic scale).

Definition at line 536 of file CoordinateSystems.cs.

7.106.4.4 MinX

```
double VectSharp.Plots.LogLinCoordinateSystem2D.MinX  [get], [set]
```

The minimum X value.

Definition at line 521 of file CoordinateSystems.cs.

7.106.4.5 MinY

```
double VectSharp.Plots.LogLinCoordinateSystem2D.MinY  [get], [set]
```

The minimum Y value (in logarithmic scale).

Definition at line 531 of file CoordinateSystems.cs.

7.106.4.6 Resolution

```
double [] VectSharp.Plots.LogLinCoordinateSystem2D.Resolution  [get], [set]
```

The maximum difference between two points in data space that appear arbitrarily close in plot space, or some approximation.

Implements VectSharp.Plots.IContinuousCoordinateSystem.

Definition at line 567 of file CoordinateSystems.cs.

7.106.4.7 ScaleX

```
double VectSharp.Plots.LogLinCoordinateSystem2D.ScaleX  [get], [set]
```

The X scale.

Definition at line 541 of file CoordinateSystems.cs.

7.106.4.8 ScaleY

```
double VectSharp.Plots.LogLinCoordinateSystem2D.ScaleY  [get], [set]
```

The y scale.

Definition at line 546 of file CoordinateSystems.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/CoordinateSystems.cs

7.107 VectSharp.Markdown.Margins Class Reference

Represents the margins of a page.

Public Member Functions

- Margins (double left, double top, double right, double bottom)

  Creates a new Margins instance.

Properties

- double Left [get]

  The left margin.
- double Right [get]

  The right margin.
- double Top [get]

  The top margin.
- double Bottom [get]

  The bottom margin.

7.107.1 Detailed Description

Represents the margins of a page.

Definition at line 299 of file MarkdownContext.cs.

7.107.2 Constructor & Destructor Documentation

7.107.2.1 Margins()

```
VectSharp.Markdown.Margins.Margins (
            double left,
            double top,
            double right,
            double bottom )
```

Creates a new Margins instance.

Parameters

left	The left margin.
top	The top margin.
right	The right margin.
bottom	The bottom margin.

Definition at line 328 of file MarkdownContext.cs.

7.107.3 Property Documentation

7.107.3.1 Bottom

```
double VectSharp.Markdown.Margins.Bottom  [get]
```

The bottom margin.

Definition at line 319 of file MarkdownContext.cs.

7.107.3.2 Left

```
double VectSharp.Markdown.Margins.Left  [get]
```

The left margin.

Definition at line 304 of file MarkdownContext.cs.

7.107.3.3 Right

```
double VectSharp.Markdown.Margins.Right  [get]
```

The right margin.

Definition at line 309 of file MarkdownContext.cs.

7.107.3.4 Top

```
double VectSharp.Markdown.Margins.Top  [get]
```

The top margin.

Definition at line 314 of file MarkdownContext.cs.

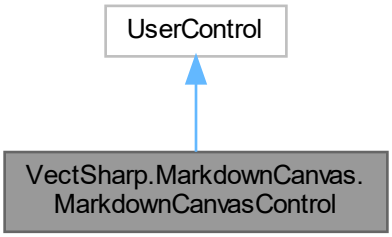
The documentation for this class was generated from the following file:

- VectSharp.Markdown/MarkdownContext.cs

7.108 VectSharp.MarkdownCanvas.MarkdownCanvasControl Class Reference

A control to display a Markdown document in an Avalonia application.

Inheritance diagram for VectSharp.MarkdownCanvas.MarkdownCanvasControl:



Public Member Functions

- MarkdownCanvasControl ()

  Initialises a new MarkdownCanvasControl.

Static Public Attributes

- static readonly StyledProperty< double > MaxRenderWidthProperty = AvaloniaProperty.Register<MarkdownCanvasControl, double>(nameof(MaxRenderWidth), double.PositiveInfinity)

  Defines the MaxRenderWidth property.
- static readonly StyledProperty< double > MinRenderWidthProperty = AvaloniaProperty.Register<MarkdownCanvasControl, double>(nameof(MinRenderWidth), 200)

  Defines the MinRenderWidth property.
- static readonly StyledProperty< double > MinVariationProperty = AvaloniaProperty.Register<MarkdownCanvasControl, double>(nameof(MinVariation), 10)

  Defines the MinVariation property.
- static readonly StyledProperty< string > DocumentSourceProperty = AvaloniaProperty.Register<MarkdownCanvasControl, string>(nameof(DocumentSource))

  Defines the DocumentSource property.
- static readonly StyledProperty< MarkdownDocument > DocumentProperty = AvaloniaProperty.↩ Register<MarkdownCanvasControl, MarkdownDocument>(nameof(Document))

  Defines the Document property.
- static readonly StyledProperty< AvaloniaContextInterpreter.TextOptions > TextConversionOptionsProperty = AvaloniaProperty.Register<MarkdownCanvasControl, AvaloniaContextInterpreter.TextOptions>(nameof(TextConversionOptior AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary)

  Defines the TextConversionOption property.

Properties

- double MaxRenderWidth [get, set]

  The maximum width for the rendered document. This will be used even if the control's client area is larger than this (the alignment of the document within the controll will depend on the control's ContentControl.HorizontalContent↩ Alignment).
- double MinRenderWidth [get, set]

  The minimum width for the rendered document. If the control's client area is smaller than this, the horizontal scroll bar will be activated.
- double MinVariation [get, set]

  The minimum width variation that triggers a document reflow. If the control is resized, but the width changes by less than this amount, the document is not re-drawn.
- string DocumentSource [set]

  Sets the currently displayed document from Markdown source.
- MarkdownDocument Document [get, set]

  Gets or sets the currently displayed MarkdownDocument.
- AvaloniaContextInterpreter.TextOptions TextConversionOption [get, set]

  Gets or sets the value that determines whether text items should be converted into paths when drawing. Setting this to AvaloniaContextInterpreter.TextOptions.NeverConvert will improve performance if you are using custom fonts, but may cause unexpected results unless the font families being used are of type ResourceFontFamily.
- MarkdownRenderer Renderer [get]

  The MarkdownRenderer used to render the Document. You can use the properties of this object to customise the rendering. Note that setting the Avalonia.Controls.Primitives.TemplatedControl.FontSize of the MarkdownCanvasControl will propagate to the Renderer's MarkdownRenderer.BaseFontSize.

7.108.1 Detailed Description

A control to display a Markdown document in an Avalonia application.

Definition at line 36 of file MarkdownCanvas.axaml.cs.

7.108.2 Constructor & Destructor Documentation

7.108.2.1 MarkdownCanvasControl()

```
VectSharp.MarkdownCanvas.MarkdownCanvasControl.MarkdownCanvasControl ( )
```

Initialises a new MarkdownCanvasControl.

Definition at line 134 of file MarkdownCanvas.axaml.cs.

7.108.3 Member Data Documentation

7.108.3.1 DocumentProperty

```
readonly StyledProperty<MarkdownDocument> VectSharp.MarkdownCanvas.MarkdownCanvasControl.↩
DocumentProperty = AvaloniaProperty.Register<MarkdownCanvasControl, MarkdownDocument>(nameof(Document))
[static]
```

Defines the Document property.

Definition at line 96 of file MarkdownCanvas.axaml.cs.

7.108.3.2 DocumentSourceProperty

```
readonly StyledProperty<string> VectSharp.MarkdownCanvas.MarkdownCanvasControl.Document↩
SourceProperty = AvaloniaProperty.Register<MarkdownCanvasControl, string>(nameof(DocumentSource))
[static]
```

Defines the DocumentSource property.

Definition at line 83 of file MarkdownCanvas.axaml.cs.

7.108.3.3 MaxRenderWidthProperty

```
readonly StyledProperty<double> VectSharp.MarkdownCanvas.MarkdownCanvasControl.MaxRender↩
WidthProperty = AvaloniaProperty.Register<MarkdownCanvasControl, double>(nameof(MaxRenderWidth),
double.PositiveInfinity)  [static]
```

Defines the MaxRenderWidth property.

Definition at line 41 of file MarkdownCanvas.axaml.cs.

7.108.3.4 MinRenderWidthProperty

```
readonly StyledProperty<double> VectSharp.MarkdownCanvas.MarkdownCanvasControl.MinRender↩
WidthProperty = AvaloniaProperty.Register<MarkdownCanvasControl, double>(nameof(MinRenderWidth),
200)  [static]
```

Defines the MinRenderWidth property.

Definition at line 55 of file MarkdownCanvas.axaml.cs.

7.108.3.5 MinVariationProperty

```
readonly StyledProperty<double> VectSharp.MarkdownCanvas.MarkdownCanvasControl.MinVariation↩
Property = AvaloniaProperty.Register<MarkdownCanvasControl, double>(nameof(MinVariation), 10)
[static]
```

Defines the MinVariation property.

Definition at line 69 of file MarkdownCanvas.axaml.cs.

7.108.3.6 TextConversionOptionsProperty

```
readonly StyledProperty<AvaloniaContextInterpreter.TextOptions> VectSharp.MarkdownCanvas.↩
MarkdownCanvasControl.TextConversionOptionsProperty = AvaloniaProperty.Register<MarkdownCanvasControl,
AvaloniaContextInterpreter.TextOptions>(nameof(TextConversionOption), AvaloniaContextInterpreter.↩
TextOptions.ConvertIfNecessary)  [static]
```

Defines the TextConversionOption property.

Definition at line 110 of file MarkdownCanvas.axaml.cs.

7.108.4 Property Documentation

7.108.4.1 Document

```
MarkdownDocument VectSharp.MarkdownCanvas.MarkdownCanvasControl.Document  [get], [set]
```

Gets or sets the currently displayed MarkdownDocument.

Definition at line 101 of file MarkdownCanvas.axaml.cs.

7.108.4.2 DocumentSource

```
string VectSharp.MarkdownCanvas.MarkdownCanvasControl.DocumentSource  [set]
```

Sets the currently displayed document from Markdown source.

Definition at line 88 of file MarkdownCanvas.axaml.cs.

7.108.4.3 MaxRenderWidth

```
double VectSharp.MarkdownCanvas.MarkdownCanvasControl.MaxRenderWidth  [get], [set]
```

The maximum width for the rendered document. This will be used even if the control's client area is larger than this (the alignment of the document within the controll will depend on the control's ContentControl.HorizontalContent↩ Alignment).

Definition at line 46 of file MarkdownCanvas.axaml.cs.

7.108.4.4 MinRenderWidth

```
double VectSharp.MarkdownCanvas.MarkdownCanvasControl.MinRenderWidth  [get], [set]
```

The minimum width for the rendered document. If the control's client area is smaller than this, the horizontal scroll bar will be activated.

Definition at line 60 of file MarkdownCanvas.axaml.cs.

7.108.4.5 MinVariation

```
double VectSharp.MarkdownCanvas.MarkdownCanvasControl.MinVariation  [get], [set]
```

The minimum width variation that triggers a document reflow. If the control is resized, but the width changes by less than this amount, the document is not re-drawn.

Definition at line 74 of file MarkdownCanvas.axaml.cs.

7.108.4.6 Renderer

```
MarkdownRenderer VectSharp.MarkdownCanvas.MarkdownCanvasControl.Renderer  [get]
```

The MarkdownRenderer used to render the Document. You can use the properties of this object to customise the rendering. Note that setting the Avalonia.Controls.Primitives.TemplatedControl.FontSize of the MarkdownCanvasControl will propagate to the Renderer's MarkdownRenderer.BaseFontSize.

Definition at line 125 of file MarkdownCanvas.axaml.cs.

7.108.4.7 TextConversionOption

```
AvaloniaContextInterpreter.TextOptions VectSharp.MarkdownCanvas.MarkdownCanvasControl.Text↩
ConversionOption [get], [set]
```

Gets or sets the value that determines whether text items should be converted into paths when drawing. Setting this to AvaloniaContextInterpreter.TextOptions.NeverConvert will improve performance if you are using custom fonts, but may cause unexpected results unless the font families being used are of type ResourceFontFamily.

Definition at line 116 of file MarkdownCanvas.axaml.cs.

The documentation for this class was generated from the following file:

- VectSharp.MarkdownCanvas/MarkdownCanvas.axaml.cs

7.109 VectSharp.Markdown.MarkdownContext Class Reference

Represents the current status of the Markdown rendering process.

Public Member Functions

- MarkdownContext Clone ()

  Create an independent copy of the current MarkdownContext.
- double GetMaxX (double y, double pageMaxX)

  Get the maximum x value for text drawn at the specified y position.
- double GetMaxX (double y0, double y1, double pageMaxX)

  Get the maximum x value for text that will extend from y0 to y1 .
- double GetMinX (double y)

  Get the minimum x value at which text can start at the specified y position.
- double GetMinX (double y0, double y1)

  Get the minimum x value for text that will extend from y0 to y1 .
- void StartNewPage (Page newPag)

  Start a new page in the document.

Properties

- Font Font `[get, set]`

  The current font.
- Point Cursor `[get, set]`

  The current position of the cursor.
- Colour Colour `[get, set]`

  The current text colour.
- bool Underline `[get, set]`

  Whether text is currently underlined.
- bool StrikeThrough `[get, set]`

  Whether text is currently struck out.
- Point Translation `[get, set]`

  Translation for the current text block.

- Point BottomRight [get, set]

  The coordinates of the bottom right corner of the last rendered text element.
- Point MarginBottomRight [get, set]

  The current margin on the bottom right.
- int ListDepth [get, set]

  The current list depth (for bullet lists and numbered lists).
- List<(double MaxX, double MinY, double MaxY)> ForbiddenAreasRight [get, set]

  A list of areas on which text cannot be drawn (because they contain a right-aligned image). Each element of this list specifies a maximum x value (first element) and a y range (second and third element). Within the specified y range, text cannot extend beyond the maximum x value.
- List<(double MinX, double MinY, double MaxY)> ForbiddenAreasLeft [get, set]

  A list of areas on which text cannot be drawn (because they contain a left-aligned image). Each element of this list specifies a minimum x value (first element) and a y range (second and third element). Within the specified y range, text starts at the minimum x value.
- Page CurrentPage [get, set]

  The Page on which the document is being rendered.
- string Tag = null [get, set]

  The current tag.
- Dictionary< string, string > LinkDestinations = new Dictionary<string, string>() [get, set]

  A dictionary used to associate graphic action tags to hyperlinks.
- Dictionary< string, string > InternalAnchors = new Dictionary<string, string>() [get, set]

  A dictionary used to associate internal html anchors (e.g., "#something") to graphics action tags.
- List<(int level, string heading, string tag)> Headings = new List<(int level, string heading, string tag)>() [get]

  The list of headings defined in the document.
- Line CurrentLine [get, set]

  The current line being drawn.

7.109.1 Detailed Description

Represents the current status of the Markdown rendering process.

Definition at line 26 of file MarkdownContext.cs.

7.109.2 Member Function Documentation

7.109.2.1 Clone()

```
MarkdownContext VectSharp.Markdown.MarkdownContext.Clone ( )
```

Create an independent copy of the current MarkdownContext.

Returns

The copied context.

Definition at line 166 of file MarkdownContext.cs.

7.109.2.2 GetMaxX() [1/2]

```
double VectSharp.Markdown.MarkdownContext.GetMaxX (
            double y,
            double pageMaxX )
```

Get the maximum x value for text drawn at the specified *y* position.

Parameters

y	The vertical position at which text should be drawn.
pageMaxX	The overall maximum x value for the page, after taking margins into account.

Returns

The maximum x value for text drawn at the specified *y* position.

Definition at line 195 of file MarkdownContext.cs.

7.109.2.3 GetMaxX() [2/2]

```
double VectSharp.Markdown.MarkdownContext.GetMaxX (
            double y0,
            double y1,
            double pageMaxX )
```

Get the maximum x value for text that will extend from *y0* to *y1* .

Parameters

y0	The vertical position of the start of the text block.
y1	The vertical position of the end of the text block.
pageMaxX	The overall maximum x value for the page, after taking margins into account.

Returns

The maximum x value for text that will extend from *y0* to *y1* .

Definition at line 219 of file MarkdownContext.cs.

7.109.2.4 GetMinX() [1/2]

```
double VectSharp.Markdown.MarkdownContext.GetMinX (
            double y )
```

Get the minimum x value at which text can start at the specified *y* position.

Parameters

y	The vertical position at which text should be drawn.

Returns

The minimum x value at which text can start at the specified *y* position.

Definition at line 242 of file MarkdownContext.cs.

7.109.2.5 GetMinX() [2/2]

```
double VectSharp.Markdown.MarkdownContext.GetMinX (
            double y0,
            double y1 )
```

Get the minimum x value for text that will extend from *y0* to *y1* .

Parameters

y0	The vertical position of the start of the text block.
y1	The vertical position of the end of the text block.

Returns

The minimum x value for text that will extend from *y0* to *y1* .

Definition at line 265 of file MarkdownContext.cs.

7.109.2.6 StartNewPage()

```
void VectSharp.Markdown.MarkdownContext.StartNewPage (
            Page newPag )
```

Start a new page in the document.

Parameters

newPag	The new page on which the rest of the document should be drawn.

Definition at line 287 of file MarkdownContext.cs.

7.109.3 Property Documentation

7.109.3.1 BottomRight

```
Point VectSharp.Markdown.MarkdownContext.BottomRight  [get], [set]
```

The coordinates of the bottom right corner of the last rendered text element.

Definition at line 97 of file MarkdownContext.cs.

7.109.3.2 Colour

```
Colour VectSharp.Markdown.MarkdownContext.Colour  [get], [set]
```

The current text colour.

Definition at line 59 of file MarkdownContext.cs.

7.109.3.3 CurrentLine

```
Line VectSharp.Markdown.MarkdownContext.CurrentLine  [get], [set]
```

The current line being drawn.

Definition at line 151 of file MarkdownContext.cs.

7.109.3.4 CurrentPage

```
Page VectSharp.Markdown.MarkdownContext.CurrentPage  [get], [set]
```

The Page on which the document is being rendered.

Definition at line 126 of file MarkdownContext.cs.

7.109.3.5 Cursor

```
Point VectSharp.Markdown.MarkdownContext.Cursor  [get], [set]
```

The current position of the cursor.

Definition at line 38 of file MarkdownContext.cs.

7.109.3.6 Font

```
Font VectSharp.Markdown.MarkdownContext.Font  [get], [set]
```

The current font.

Definition at line 31 of file MarkdownContext.cs.

7.109.3.7 ForbiddenAreasLeft

```
List<(double MinX, double MinY, double MaxY)> VectSharp.Markdown.MarkdownContext.Forbidden↩
AreasLeft  [get], [set]
```

A list of areas on which text cannot be drawn (because they contain a left-aligned image). Each element of this list specifies a minimum x value (first element) and a y range (second and third element). Within the specified y range, text starts at the minimum x value.

Definition at line 121 of file MarkdownContext.cs.

7.109.3.8 ForbiddenAreasRight

```
List<(double MaxX, double MinY, double MaxY)> VectSharp.Markdown.MarkdownContext.Forbidden↩
AreasRight  [get], [set]
```

A list of areas on which text cannot be drawn (because they contain a right-aligned image). Each element of this list specifies a maximum x value (first element) and a y range (second and third element). Within the specified y range, text cannot extend beyond the maximum x value.

Definition at line 114 of file MarkdownContext.cs.

7.109.3.9 Headings

```
List<(int level, string heading, string tag)> VectSharp.Markdown.MarkdownContext.Headings =
new List<(int level, string heading, string tag)>()  [get]
```

The list of headings defined in the document.

Definition at line 146 of file MarkdownContext.cs.

7.109.3.10 InternalAnchors

```
Dictionary<string, string> VectSharp.Markdown.MarkdownContext.InternalAnchors = new Dictionary<string,
string>()  [get], [set]
```

A dictionary used to associate internal html anchors (e.g., "#something") to graphics action tags.

Definition at line 141 of file MarkdownContext.cs.

7.109.3.11 LinkDestinations

```
Dictionary<string, string> VectSharp.Markdown.MarkdownContext.LinkDestinations = new Dictionary<string,
string>()  [get], [set]
```

A dictionary used to associate graphic action tags to hyperlinks.

Definition at line 136 of file MarkdownContext.cs.

7.109.3.12 ListDepth

```
int VectSharp.Markdown.MarkdownContext.ListDepth  [get], [set]
```

The current list depth (for bullet lists and numbered lists).

Definition at line 107 of file MarkdownContext.cs.

7.109.3.13 MarginBottomRight

```
Point VectSharp.Markdown.MarkdownContext.MarginBottomRight  [get], [set]
```

The current margin on the bottom right.

Definition at line 102 of file MarkdownContext.cs.

7.109.3.14 StrikeThrough

```
bool VectSharp.Markdown.MarkdownContext.StrikeThrough  [get], [set]
```

Whether text is currently struck out.

Definition at line 69 of file MarkdownContext.cs.

7.109.3.15 Tag

```
string VectSharp.Markdown.MarkdownContext.Tag = null  [get], [set]
```

The current tag.

Definition at line 131 of file MarkdownContext.cs.

7.109.3.16 Translation

```
Point VectSharp.Markdown.MarkdownContext.Translation  [get], [set]
```

Translation for the current text block.

Definition at line 76 of file MarkdownContext.cs.

7.109.3.17 Underline

```
bool VectSharp.Markdown.MarkdownContext.Underline  [get], [set]
```

Whether text is currently underlined.

Definition at line 64 of file MarkdownContext.cs.

The documentation for this class was generated from the following file:

- VectSharp.Markdown/MarkdownContext.cs

7.110 VectSharp.Markdown.MarkdownEmojiExtension Class Reference

Contains extension methods for the MarkdownPipelineBuilder class.

Static Public Member Functions

- static MarkdownPipelineBuilder UseUnicodeEmoji (this MarkdownPipelineBuilder pipeline)
  Uses the extension enabling unicode emoji resolution.

7.110.1 Detailed Description

Contains extension methods for the MarkdownPipelineBuilder class.

Definition at line 32 of file UnicodeEmojiParser.cs.

7.110.2 Member Function Documentation

7.110.2.1 UseUnicodeEmoji()

```
static MarkdownPipelineBuilder VectSharp.Markdown.MarkdownEmojiExtension.UseUnicodeEmoji (
            this MarkdownPipelineBuilder pipeline )  [static]
```

Uses the extension enabling unicode emoji resolution.

Parameters

pipeline	The pipeline.

Returns

The modified pipeline.

Definition at line 39 of file UnicodeEmojiParser.cs.

The documentation for this class was generated from the following file:

- VectSharp.Markdown/UnicodeEmojiParser.cs

7.111 VectSharp.Markdown.MarkdownRenderer Class Reference

Renders Markdown documents into VectSharp graphics objects.

Public Types

- enum VerticalAlignment

  Defines the options for the vertical alignment of table cells.

Public Member Functions

- MarkdownRenderer ()

  Create a new MarkdownRenderer;
- Page RenderSinglePage (string markdownSource, double width, out Dictionary< string, string > linkDestinations, out List<(int level, string heading, string tag)> headingTree)

  Parses the supplied markdownSource using all the supported extensions and renders the resulting document. Page breaks are disabled, and the document is rendered as a single page with the specified width . The page will be cropped at the appropriate height to contain the entire document.
- Page RenderSinglePage (MarkdownDocument markdownDocument, double width, out Dictionary< string, string > linkDestinations, out List<(int level, string heading, string tag)> headingTree)

  Renders the supplied markdownDocument . Page breaks are disabled, and the document is rendered as a single page with the specified width . The page will be cropped at the appropriate height to contain the entire document.
- Document Render (string markdownSource, out Dictionary< string, string > linkDestinations, out List<(int level, string heading, string tag)> headingTree)

  Parses the supplied markdownSource using all the supported extensions and renders the resulting document. The Document produced consists of one or more pages of the size specified in the PageSize of the current instance.
- Document Render (MarkdownDocument mardownDocument, out Dictionary< string, string > linkDestinations, out List<(int level, string heading, string tag)> headingTree)

  Renders the supplied mardownDocument . The Document produced consists of one or more pages of the size specified in the PageSize of the current instance.

Properties

- double BaseFontSize = 9.71424 [get, set]

  The base font size to use when rendering the document. This will be the size of regular elements, and the size of header elements will be expressed as a multiple of this.
- double MathFontScalingFactor = 0.85 [get, set]

  Scaling factor for the font size to use when rendering math.
- double[ ] HeaderFontSizeMultipliers [get]

  The font size for elements at each header level. The values in this array will be multiplied by the BaseFontSize.
- double[ ] HeaderLineThicknesses = new double[ ] { 1, 1, 0, 0, 0, 0 } [get]

  The thickness of the separator line after a header of each level. A value of 0 disables the line after headers of that level.
- double ThematicBreakThickness = 2 [get, set]

  The thickness of thematic break lines.
- FontFamily RegularFontFamily = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica) [get, set]

  The font family for regular text.
- FontFamily BoldFontFamily = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica↩Bold) [get, set]

  The font family for bold text.
- FontFamily ItalicFontFamily = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica↩Oblique) [get, set]

  The font family for italic text.
- FontFamily BoldItalicFontFamily = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.↩HelveticaBoldOblique) [get, set]

  The font family for bold italic text.
- FontFamily CodeFont = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Courier) [get, set]

  The font family for code elements.
- FontFamily CodeFontBold = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierBold) [get, set]

  The font family for bold code elements.
- FontFamily CodeFontItalic = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Courier↩Oblique) [get, set]

  The font family for italic code elements.
- FontFamily CodeFontBoldItalic = FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Courier↩BoldOblique) [get, set]

  The font family for bold italic code elements.
- double UnderlineThickness = 0.075 [get, set]

  The thickness of underlines. This value will be multiplied by the font size of the element being underlined.
- double BoldUnderlineThickness = 0.15 [get, set]

  The thickness of underlines for bold text. This value will be multiplied by the font size of the element being underlined.
- Margins Margins = new Margins(55, 55, 55, 55) [get, set]

  The margins of the page.
- Margins TableCellMargins = new Margins(5, 0, 5, 0) [get, set]

  The margins for table cells.
- VerticalAlignment TableVAlign = VerticalAlignment.Middle [get, set]

  The vertical alignment of table cells.
- Size PageSize = new Size(595, 842) [get, set]

  The size of the page.
- double SpaceBeforeParagaph = 0 [get, set]

  The space before each text paragraph. This value will be multiplied by the BaseFontSize.

- double SpaceAfterParagraph = 0.75 [get, set]

  The space after each text paragraph. This value will be multiplied by the BaseFontSize.
- double SpaceAfterLine = 0.25 [get, set]

  The space after each line of text. This value will be multiplied by the BaseFontSize.
- double SpaceBeforeHeading = 0.25 [get, set]

  The space before each heading. This value will be multiplied by the font size of the heading.
- double SpaceAfterHeading = 0.25 [get, set]

  The space after each heading. This value will be multiplied by the font size of the heading.
- double CodeInlineMargin = 0.25 [get, set]

  The margin at the left and right of code inlines. This value will be multiplied by the current font size.
- double IndentWidth = 40 [get, set]

  The indentation width used for list items.
- double QuoteBlockIndentWidth = 30 [get, set]

  The indentation width used for block quotes.
- double QuoteBlockBarWidth = 5 [get, set]

  The thickness of the bar to the left of block quotes.
- double SubSuperscriptFontSize = 0.7 [get, set]

  The font size for subscripts and superscripts. This value will be multiplied by the current font size.
- double SuperscriptShift = 0.33 [get, set]

  The upwards shift in the baseline for superscript elements. This value will be multiplied by the current font size.
- double SubscriptShift = 0.14 [get, set]

  The downwards shift in the baseline for subscript elements. This value will be multiplied by the current font size.
- string BaseImageUri = "" [get, set]

  The base uri for resolving relative image addresses.
- Func< string, string,(string, bool)> ImageUriResolver = HTTPUtils.ResolveImageURI [get, set]

  A method used to resolve (possibly remote) image uris into local file paths. The first argument of the method should be the image uri and the second argument the base uri used to resolve relative links. The method should return a tuple containing the path of the local file and a boolean value indicating whether the file has been fetched from a remote location and should be deleted after the program has finished using it.
- Func< string,(string, bool)> EmojiUriResolver [get, set]

  A method used to resolve emojis. The argument of the method should be an emoji uri of the form "emoji://{name}_↩ heading:{level}", where "{name}" is the name of the emoji and "{level}" is the heading level, or "unicode://{unicode}_↩ heading:{level}", where "{unicode}" is a unicode surrogate pair. The method should return a tuple containing the path of a local file containing the rendered emoji and a boolean value indicating whether the file should be deleted after the program has finished using it.
- Uri BaseLinkUri = new Uri("about:blank") [get, set]

  The base uri for resolving links.
- Func< string, string > LinkUriResolver = a => a [get, set]

  A method used to resolve link addresses. The argument of the method should be the absolute link, and the method should return the resolved address. This can be used to "redirect" links to a different target.
- Func< string, RasterImage > RasterImageLoader = null [get, set]

  A method used to a load raster image from a local file. The argument of the method should be the path of a local image file, and the method should return a RasterImage representing that file. For example, this can be achieved using the `RasterImageFile` class from the `VectSharp.MuPDFUtils` package. If this is `null`, only SVG images will be included in the document.
- double ImageUnitMultiplier = 0.60714 [get, set]

  The size of images (as defined in the image's width and height attributes) will be multiplied by this value to determine the actual size of the image on the page. This has no effect on images without a width or height attribute.
- double ImageMultiplier = 1 [get, set]

  The size of images will be multiplied by this value to determine the actual size of the image on the page. For images that have a width or height attribute, this will be applied in addition to the ImageUnitMultiplier. For images without width and height, only this multiplier will be applied.
- double ImageSideMargin = 10 [get, set]

The margin on the right of left-aligned images and on the left of right-aligned images.

- double ImageMarginTolerance = 25 `[get, set]`

  Images will be allowed to extend into the page bottom margin area by this amount before triggering a page break. This should be smaller than the bottom margin, otherwise images risk being cut off by the page boundary.

- Func< string, string, List< List< FormattedString > > > SyntaxHighlighter = VectSharp.Markdown.SyntaxHighlighter.GetSynta `[get, set]`

  A method used for syntax highlighting. The first argument should be the source code to highlight, while the second parameter is the name of the language to use for the highlight. The method should return a list of lists of FormattedStrings, with each list of FormattedStrings representing a line. For each code block, if the method returns `null`, no syntax highlighting is used.

- List< Action< Graphics, Colour > > Bullets `[get]`

  Bullet points used for unordered lists. Each element of this list corresponds to the bullet for each level of list indentation. If the list indentation is greater than the number of elements in this list, the bullet points will be reused cyclically. Each element of this list is a method taking two arguments: the first is the Graphics object on which the bullet point should be drawn, while the second is the colour in which it should be painted. The method should draw the bullet point centered around the origin. The size of the bullet point will be multiplied by the current font size.

- Graphics TaskListUncheckedBullet `[get, set]`

  The bullet used for unchecked task list items.

- Graphics TaskListCheckedBullet `[get, set]`

  The bullet used for checked task list items.

- Colour ForegroundColour = Colours.Black `[get, set]`

  The foreground colour for text elements.

- Colour BackgroundColour = Colours.White `[get, set]`

  The background colour for the page.

- Colour HeaderLineColour = Colour.FromRgb(180, 180, 180) `[get, set]`

  The colour of the line below headers.

- Colour ThematicBreakLineColour = Colour.FromRgb(180, 180, 200) `[get, set]`

  The colour for thematic break lines.

- Colour LinkColour = Colour.FromRgb(25, 140, 191) `[get, set]`

  The colour for hypertext links.

- Colour CodeInlineBackgroundColour = Colour.FromRgb(240, 240, 240) `[get, set]`

  The background colour for code inlines.

- Colour CodeBlockBackgroundColour = Colour.FromRgb(240, 240, 245) `[get, set]`

  The background colour for code blocks.

- Colour QuoteBlockBarColour = Colour.FromRgb(75, 152, 220) `[get, set]`

  The colour for the bar to the left of block quotes.

- Colour QuoteBlockBackgroundColour = Colour.FromRgb(240, 240, 255) `[get, set]`

  The background colour for block quotes.

- Colour InsertedColour = Colour.FromRgb(0, 158, 115) `[get, set]`

  The colour for text that has been styled as "inserted".

- Colour MarkedColour = Colour.FromRgb(213, 94, 0) `[get, set]`

  The colour for text that has been styled as "marked".

- Colour TableHeaderRowSeparatorColour = Colours.Black `[get, set]`

  The colour for the line separating the table header row from normal rows.

- Colour TableRowSeparatorColour = Colour.FromRgb(180, 180, 180) `[get, set]`

  The colour for lines separating table rows from each other.

- double TableHeaderRowSeparatorThickness = 2 `[get, set]`

  The thickness of the line separating the table header row from normal rows.

- double TableHeaderSeparatorThickness = 1 `[get, set]`

  The thickness of lines separating table rows from each other.

- bool AllowPageBreak = true `[get, set]`

  Determines whether page breaks should be treated as such in the source.

- static EmojiMapping EmojiURLMapping = new EmojiMapping(new Dictionary<string, string>(EmojiMapping.GetDefaultEmojiShortcodeToUnicode().Select(x => new KeyValuePair<string, string>(x.Key, "emoji://" + x.Key.Trim(':')))), EmojiMapping.GetDefaultSmileyToEmojiShortcode()) [get]

  Emoji mapping that transforms emojis (e.g. ":something:") into URLs (e.g., "emoji://something").
- MarkdownPipelineBuilder MarkdownPipelineBuilder = new MarkdownPipelineBuilder().UseGridTables().UsePipeTables().UseEmphasisExtras().UseGenericAttributes().UseAutoIdentifiers().UseAutoLinks().UseTaskLists().UseListExtras().UseCitations().UseMathematics().UseSmartyPants().UseEmojiAndSmiley(EmojiURLMapping).UseUnicodeEmoji() [get, set]

  Markdown pipeline builder used to parse markdown source.

Events

- EventHandler< PageStartedEventArgs > PageStarted

  Invoked when a new page is started.
- EventHandler< PageFinishedEventArgs > PageFinished

  Invoked when a page is finished, before starting a new page (or at the end of the document).
- EventHandler< BlockRenderingEventArgs > BlockRendering

  Invoked just before rendering a Block.
- EventHandler< BlockRenderedEventArgs > BlockRendered

  Invoked just after rendering a Block.
- EventHandler< InlineRenderingEventArgs > InlineRendering

  Invoked just before rendering an Inline.
- EventHandler< InlineRenderedEventArgs > InlineRendered

  Invoked just after rendering an Inline.
- EventHandler< LineMeasuredEventArgs > LineMeasured

  Invoked after content in a Line has been measured, but before deciding whether to render the line on the current page or on a new page.
- EventHandler< LineEventArgs > LineRendering

  Invoked just before rendering a Line, after a new page has been created if necessary.
- EventHandler< LineEventArgs > LineRendered

  Invoked just after rendering a Line.

7.111.1 Detailed Description

Renders Markdown documents into VectSharp graphics objects.

Definition at line 32 of file MarkdownRenderer.Blocks.cs.

7.111.2 Member Enumeration Documentation

7.111.2.1 VerticalAlignment

enum VectSharp.Markdown.MarkdownRenderer.VerticalAlignment

Defines the options for the vertical alignment of table cells.

Definition at line 119 of file MarkdownRenderer.Properties.cs.

7.111.3 Constructor & Destructor Documentation

7.111.3.1 MarkdownRenderer()

```
VectSharp.Markdown.MarkdownRenderer.MarkdownRenderer ( )
```

Create a new MarkdownRenderer;

Definition at line 44 of file MarkdownRenderer.cs.

7.111.4 Member Function Documentation

7.111.4.1 Render() [1/2]

```
Document VectSharp.Markdown.MarkdownRenderer.Render (
            MarkdownDocument mardownDocument,
            out Dictionary< string, string > linkDestinations,
            out List<(int level, string heading, string tag)> headingTree )
```

Renders the supplied *mardownDocument* . The Document produced consists of one or more pages of the size specified in the PageSize of the current instance.

Parameters

mardownDocument	The markdown document to render.
linkDestinations	When this method returns, this object will contain a dictionary used to associate graphic action tags to hyperlinks. This can be used to enable such links when rendering the Document to a file.
headingTree	When this method returns, this object will contain a list of the headings contained in the document and the corresponding tag. This can be used to create a document outline.

Returns

A Document containing a rendering of the supplied markdown document, consisting of one or more pages of the size specified in the PageSize of the current instance.

Definition at line 157 of file MarkdownRenderer.cs.

7.111.4.2 Render() [2/2]

```
Document VectSharp.Markdown.MarkdownRenderer.Render (
            string markdownSource,
```

```
            out Dictionary< string, string > linkDestinations,
            out List<(int level, string heading, string tag)> headingTree )
```

Parses the supplied *markdownSource* using all the supported extensions and renders the resulting document. The Document produced consists of one or more pages of the size specified in the PageSize of the current instance.

Parameters

markdownSource	The markdown source to parse.
linkDestinations	When this method returns, this value will contain a dictionary used to associate graphic action tags to hyperlinks. This can be used to enable such links when rendering the Document to a file.
headingTree	When this method returns, this object will contain a list of the headings contained in the document and the corresponding tag. This can be used to create a document outline.

Returns

A Document containing a rendering of the supplied markdown document, consisting of one or more pages of the size specified in the PageSize of the current instance.

Definition at line 143 of file MarkdownRenderer.cs.

7.111.4.3 RenderSinglePage() [1/2]

```
Page VectSharp.Markdown.MarkdownRenderer.RenderSinglePage (
            MarkdownDocument markdownDocument,
            double width,
            out Dictionary< string, string > linkDestinations,
            out List<(int level, string heading, string tag)> headingTree )
```

Renders the supplied *markdownDocument* . Page breaks are disabled, and the document is rendered as a single page with the specified *width* . The page will be cropped at the appropriate height to contain the entire document.

Parameters

markdownDocument	The markdown document to render.
width	The width of the page.
linkDestinations	When this method returns, this value will contain a dictionary used to associate graphic action tags to hyperlinks. This can be used to enable such links when rendering the Page to a file.
headingTree	When this method returns, this object will contain a list of the headings contained in the document and the corresponding tag. This can be used to create a document outline.

Returns

A Page containing a rendering of the supplied markdown document.

Definition at line 81 of file MarkdownRenderer.cs.

7.111.4.4 RenderSinglePage() [2/2]

```
Page VectSharp.Markdown.MarkdownRenderer.RenderSinglePage (
            string markdownSource,
            double width,
            out Dictionary< string, string > linkDestinations,
            out List<(int level, string heading, string tag)> headingTree )
```

Parses the supplied *markdownSource* using all the supported extensions and renders the resulting document. Page breaks are disabled, and the document is rendered as a single page with the specified *width* . The page will be cropped at the appropriate height to contain the entire document.

Parameters

markdownSource	The markdown source to parse.
width	The width of the page.
linkDestinations	When this method returns, this value will contain a dictionary used to associate graphic action tags to hyperlinks. This can be used to enable such links when rendering the Page to a file.
headingTree	When this method returns, this object will contain a list of the headings contained in the document and the corresponding tag. This can be used to create a document outline.

Returns

A Page containing a rendering of the supplied markdown document.

Definition at line 66 of file MarkdownRenderer.cs.

7.111.5 Property Documentation

7.111.5.1 AllowPageBreak

```
bool VectSharp.Markdown.MarkdownRenderer.AllowPageBreak = true  [get], [set]
```

Determines whether page breaks should be treated as such in the source.

Definition at line 395 of file MarkdownRenderer.Properties.cs.

7.111.5.2 BackgroundColour

```
Colour VectSharp.Markdown.MarkdownRenderer.BackgroundColour = Colours.White  [get], [set]
```

The background colour for the page.

Definition at line 325 of file MarkdownRenderer.Properties.cs.

7.111.5.3 BaseFontSize

```
double VectSharp.Markdown.MarkdownRenderer.BaseFontSize = 9.71424  [get], [set]
```

The base font size to use when rendering the document. This will be the size of regular elements, and the size of header elements will be expressed as a multiple of this.

Definition at line 31 of file MarkdownRenderer.Properties.cs.

7.111.5.4 BaseImageUri

```
string VectSharp.Markdown.MarkdownRenderer.BaseImageUri = ""  [get], [set]
```

The base uri for resolving relative image addresses.

Definition at line 210 of file MarkdownRenderer.Properties.cs.

7.111.5.5 BaseLinkUri

```
Uri VectSharp.Markdown.MarkdownRenderer.BaseLinkUri = new Uri("about:blank")  [get], [set]
```

The base uri for resolving links.

Definition at line 225 of file MarkdownRenderer.Properties.cs.

7.111.5.6 BoldFontFamily

```
FontFamily VectSharp.Markdown.MarkdownRenderer.BoldFontFamily = FontFamily.ResolveFontFamily(Font↩
Family.StandardFontFamilies.HelveticaBold)  [get], [set]
```

The font family for bold text.

Definition at line 64 of file MarkdownRenderer.Properties.cs.

7.111.5.7 BoldItalicFontFamily

```
FontFamily VectSharp.Markdown.MarkdownRenderer.BoldItalicFontFamily = FontFamily.ResolveFontFamily(Font↩
Family.StandardFontFamilies.HelveticaBoldOblique)  [get], [set]
```

The font family for bold italic text.

Definition at line 74 of file MarkdownRenderer.Properties.cs.

7.111.5.8 BoldUnderlineThickness

```
double VectSharp.Markdown.MarkdownRenderer.BoldUnderlineThickness = 0.15  [get], [set]
```

The thickness of underlines for bold text. This value will be multiplied by the font size of the element being underlined.

Definition at line 104 of file MarkdownRenderer.Properties.cs.

7.111.5.9 Bullets

```
List<Action<Graphics, Colour> > VectSharp.Markdown.MarkdownRenderer.Bullets  [get]
```

Initial value:

```
= new List<Action<Graphics, Colour»()
        {
            (graphics, colour) =>
            {
                graphics.FillPath(new GraphicsPath().Arc(-0.5, 0, 0.25, 0, 2 * Math.PI), colour);
            },
            (graphics, colour) =>
            {
                graphics.StrokePath(new GraphicsPath().Arc(-0.5, 0, 0.25, 0, 2 * Math.PI), colour, 0.1);
            },
            (graphics, colour) =>
            {
                graphics.StrokeRectangle(-0.75, -0.25, 0.5, 0.5, colour, 0.1);
            },
        }
```

Bullet points used for unordered lists. Each element of this list corresponds to the bullet for each level of list indentation. If the list indentation is greater than the number of elements in this list, the bullet points will be reused cyclically. Each element of this list is a method taking two arguments: the first is the Graphics object on which the bullet point should be drawn, while the second is the colour in which it should be painted. The method should draw the bullet point centered around the origin. The size of the bullet point will be multiplied by the current font size.

Definition at line 266 of file MarkdownRenderer.Properties.cs.

7.111.5.10 CodeBlockBackgroundColour

```
Colour VectSharp.Markdown.MarkdownRenderer.CodeBlockBackgroundColour = Colour.FromRgb(240,
240, 245)  [get], [set]
```

The background colour for code blocks.

Definition at line 350 of file MarkdownRenderer.Properties.cs.

7.111.5.11 CodeFont

```
FontFamily VectSharp.Markdown.MarkdownRenderer.CodeFont = FontFamily.ResolveFontFamily(Font↩
Family.StandardFontFamilies.Courier)  [get], [set]
```

The font family for code elements.

Definition at line 79 of file MarkdownRenderer.Properties.cs.

7.111.5.12 CodeFontBold

```
FontFamily VectSharp.Markdown.MarkdownRenderer.CodeFontBold = FontFamily.ResolveFontFamily(Font↩
Family.StandardFontFamilies.CourierBold)  [get], [set]
```

The font family for bold code elements.

Definition at line 84 of file MarkdownRenderer.Properties.cs.

7.111.5.13 CodeFontBoldItalic

```
FontFamily VectSharp.Markdown.MarkdownRenderer.CodeFontBoldItalic = FontFamily.ResolveFontFamily(Font↩
Family.StandardFontFamilies.CourierBoldOblique)  [get], [set]
```

The font family for bold italic code elements.

Definition at line 94 of file MarkdownRenderer.Properties.cs.

7.111.5.14 CodeFontItalic

```
FontFamily VectSharp.Markdown.MarkdownRenderer.CodeFontItalic = FontFamily.ResolveFontFamily(Font↩
Family.StandardFontFamilies.CourierOblique)  [get], [set]
```

The font family for italic code elements.

Definition at line 89 of file MarkdownRenderer.Properties.cs.

7.111.5.15 CodeInlineBackgroundColour

```
Colour VectSharp.Markdown.MarkdownRenderer.CodeInlineBackgroundColour = Colour.FromRgb(240,
240, 240)  [get], [set]
```

The background colour for code inlines.

Definition at line 345 of file MarkdownRenderer.Properties.cs.

7.111.5.16 CodeInlineMargin

```
double VectSharp.Markdown.MarkdownRenderer.CodeInlineMargin = 0.25  [get], [set]
```

The margin at the left and right of code inlines. This value will be multiplied by the current font size.

Definition at line 175 of file MarkdownRenderer.Properties.cs.

7.111.5.17 EmojiUriResolver

```
Func<string, (string, bool)> VectSharp.Markdown.MarkdownRenderer.EmojiUriResolver [get],
[set]
```

A method used to resolve emojis. The argument of the method should be an emoji uri of the form "emoji://{name}_↩
heading:{level}", where "{name}" is the name of the emoji and "{level}" is the heading level, or "unicode://{unicode}↩
_heading:{level}", where "{unicode}" is a unicode surrogate pair. The method should return a tuple containing the path of a local file containing the rendered emoji and a boolean value indicating whether the file should be deleted after the program has finished using it.

Definition at line 220 of file MarkdownRenderer.Properties.cs.

7.111.5.18 EmojiURLMapping

```
EmojiMapping VectSharp.Markdown.MarkdownRenderer.EmojiURLMapping = new EmojiMapping(new Dictionary<string,
string>(EmojiMapping.GetDefaultEmojiShortcodeToUnicode().Select(x => new KeyValuePair<string,
string>(x.Key, "emoji://" + x.Key.Trim(':')))), EmojiMapping.GetDefaultSmileyToEmojiShortcode())
[static], [get]
```

Emoji mapping that transforms emojis (e.g. ":something:") into URLs (e.g., "emoji://something").

Definition at line 400 of file MarkdownRenderer.Properties.cs.

7.111.5.19 ForegroundColour

```
Colour VectSharp.Markdown.MarkdownRenderer.ForegroundColour = Colours.Black [get], [set]
```

The foreground colour for text elements.

Definition at line 320 of file MarkdownRenderer.Properties.cs.

7.111.5.20 HeaderFontSizeMultipliers

```
double [] VectSharp.Markdown.MarkdownRenderer.HeaderFontSizeMultipliers [get]
```

Initial value:

```
= new double[]
        {
            28 / 12.0, 22 / 12.0, 16 / 12.0, 14 / 12.0, 13 / 12.0, 12 / 12.0
        }
```

The font size for elements at each header level. The values in this array will be multiplied by the BaseFontSize.

Definition at line 41 of file MarkdownRenderer.Properties.cs.

7.111.5.21 HeaderLineColour

```
Colour VectSharp.Markdown.MarkdownRenderer.HeaderLineColour = Colour.FromRgb(180, 180, 180)
[get], [set]
```

The colour of the line below headers.

Definition at line 330 of file MarkdownRenderer.Properties.cs.

7.111.5.22 HeaderLineThicknesses

```
double [] VectSharp.Markdown.MarkdownRenderer.HeaderLineThicknesses = new double[] { 1, 1, 0,
0, 0, 0 }  [get]
```

The thickness of the separator line after a header of each level. A value of 0 disables the line after headers of that level.

Definition at line 49 of file MarkdownRenderer.Properties.cs.

7.111.5.23 ImageMarginTolerance

```
double VectSharp.Markdown.MarkdownRenderer.ImageMarginTolerance = 25  [get], [set]
```

Images will be allowed to extend into the page bottom margin area by this amount before triggering a page break. This should be smaller than the bottom margin, otherwise images risk being cut off by the page boundary.

Definition at line 255 of file MarkdownRenderer.Properties.cs.

7.111.5.24 ImageMultiplier

```
double VectSharp.Markdown.MarkdownRenderer.ImageMultiplier = 1  [get], [set]
```

The size of images will be multiplied by this value to determine the actual size of the image on the page. For images that have a width or height attribute, this will be applied in addition to the ImageUnitMultiplier. For images without width and height, only this multiplier will be applied.

Definition at line 245 of file MarkdownRenderer.Properties.cs.

7.111.5.25 ImageSideMargin

```
double VectSharp.Markdown.MarkdownRenderer.ImageSideMargin = 10  [get], [set]
```

The margin on the right of left-aligned images and on the left of right-aligned images.

Definition at line 250 of file MarkdownRenderer.Properties.cs.

7.111.5.26 ImageUnitMultiplier

```
double VectSharp.Markdown.MarkdownRenderer.ImageUnitMultiplier = 0.60714  [get], [set]
```

The size of images (as defined in the image's width and height attributes) will be multiplied by this value to determine the actual size of the image on the page. This has no effect on images without a width or height attribute.

Definition at line 240 of file MarkdownRenderer.Properties.cs.

7.111.5.27 ImageUriResolver

```
Func<string, string, (string, bool)> VectSharp.Markdown.MarkdownRenderer.ImageUriResolver =
HTTPUtils.ResolveImageURI  [get], [set]
```

A method used to resolve (possibly remote) image uris into local file paths. The first argument of the method should be the image uri and the second argument the base uri used to resolve relative links. The method should return a tuple containing the path of the local file and a boolean value indicating whether the file has been fetched from a remote location and should be deleted after the program has finished using it.

Definition at line 215 of file MarkdownRenderer.Properties.cs.

7.111.5.28 IndentWidth

```
double VectSharp.Markdown.MarkdownRenderer.IndentWidth = 40  [get], [set]
```

The indentation width used for list items.

Definition at line 180 of file MarkdownRenderer.Properties.cs.

7.111.5.29 InsertedColour

```
Colour VectSharp.Markdown.MarkdownRenderer.InsertedColour = Colour.FromRgb(0, 158, 115)  [get],
[set]
```

The colour for text that has been styled as "inserted".

Definition at line 365 of file MarkdownRenderer.Properties.cs.

7.111.5.30 ItalicFontFamily

```
FontFamily VectSharp.Markdown.MarkdownRenderer.ItalicFontFamily = FontFamily.ResolveFontFamily(Font↩
Family.StandardFontFamilies.HelveticaOblique)  [get], [set]
```

The font family for italic text.

Definition at line 69 of file MarkdownRenderer.Properties.cs.

7.111.5.31 LinkColour

```
Colour VectSharp.Markdown.MarkdownRenderer.LinkColour = Colour.FromRgb(25, 140, 191)  [get],
[set]
```

The colour for hypertext links.

Definition at line 340 of file MarkdownRenderer.Properties.cs.

7.111.5.32 LinkUriResolver

```
Func<string, string> VectSharp.Markdown.MarkdownRenderer.LinkUriResolver = a => a  [get],
[set]
```

A method used to resolve link addresses. The argument of the method should be the absolute link, and the method should return the resolved address. This can be used to "redirect" links to a different target.

Definition at line 230 of file MarkdownRenderer.Properties.cs.

7.111.5.33 Margins

```
Margins VectSharp.Markdown.MarkdownRenderer.Margins = new Margins(55, 55, 55, 55)  [get], [set]
```

The margins of the page.

Definition at line 109 of file MarkdownRenderer.Properties.cs.

7.111.5.34 MarkdownPipelineBuilder

```
MarkdownPipelineBuilder VectSharp.Markdown.MarkdownRenderer.MarkdownPipelineBuilder = new
MarkdownPipelineBuilder().UseGridTables().UsePipeTables().UseEmphasisExtras().UseGeneric↩
Attributes().UseAutoIdentifiers().UseAutoLinks().UseTaskLists().UseListExtras().UseCitations().Use↩
Mathematics().UseSmartyPants().UseEmojiAndSmiley(EmojiURLMapping).UseUnicodeEmoji()  [get],
[set]
```

Markdown pipeline builder used to parse markdown source.

Definition at line 405 of file MarkdownRenderer.Properties.cs.

7.111.5.35 MarkedColour

```
Colour VectSharp.Markdown.MarkdownRenderer.MarkedColour = Colour.FromRgb(213, 94, 0)  [get], [set]
```

The colour for text that has been styled as "marked".

Definition at line 370 of file MarkdownRenderer.Properties.cs.

7.111.5.36 MathFontScalingFactor

```
double VectSharp.Markdown.MarkdownRenderer.MathFontScalingFactor = 0.85  [get], [set]
```

Scaling factor for the font size to use when rendering math.

Definition at line 36 of file MarkdownRenderer.Properties.cs.

7.111.5.37 PageSize

```
Size VectSharp.Markdown.MarkdownRenderer.PageSize = new Size(595, 842)  [get], [set]
```

The size of the page.

Definition at line 145 of file MarkdownRenderer.Properties.cs.

7.111.5.38 QuoteBlockBackgroundColour

```
Colour VectSharp.Markdown.MarkdownRenderer.QuoteBlockBackgroundColour = Colour.FromRgb(240, 240, 255)  [get], [set]
```

The background colour for block quotes.

Definition at line 360 of file MarkdownRenderer.Properties.cs.

7.111.5.39 QuoteBlockBarColour

```
Colour VectSharp.Markdown.MarkdownRenderer.QuoteBlockBarColour = Colour.FromRgb(75, 152, 220) [get], [set]
```

The colour for the bar to the left of block quotes.

Definition at line 355 of file MarkdownRenderer.Properties.cs.

7.111.5.40 QuoteBlockBarWidth

```
double VectSharp.Markdown.MarkdownRenderer.QuoteBlockBarWidth = 5  [get], [set]
```

The thickness of the bar to the left of block quotes.

Definition at line 190 of file MarkdownRenderer.Properties.cs.

7.111.5.41 QuoteBlockIndentWidth

```
double VectSharp.Markdown.MarkdownRenderer.QuoteBlockIndentWidth = 30  [get], [set]
```

The indentation width used for block quotes.

Definition at line 185 of file MarkdownRenderer.Properties.cs.

7.111.5.42 RasterImageLoader

```
Func<string, RasterImage> VectSharp.Markdown.MarkdownRenderer.RasterImageLoader = null  [get],
[set]
```

A method used to a load raster image from a local file. The argument of the method should be the path of a local image file, and the method should return a RasterImage representing that file. For example, this can be achieved using the `RasterImageFile` class from the `VectSharp.MuPDFUtils` package. If this is `null`, only SVG images will be included in the document.

Definition at line 235 of file MarkdownRenderer.Properties.cs.

7.111.5.43 RegularFontFamily

```
FontFamily VectSharp.Markdown.MarkdownRenderer.RegularFontFamily = FontFamily.ResolveFontFamily(Font↩
Family.StandardFontFamilies.Helvetica)  [get], [set]
```

The font family for regular text.

Definition at line 59 of file MarkdownRenderer.Properties.cs.

7.111.5.44 SpaceAfterHeading

```
double VectSharp.Markdown.MarkdownRenderer.SpaceAfterHeading = 0.25  [get], [set]
```

The space after each heading. This value will be multiplied by the font size of the heading.

Definition at line 170 of file MarkdownRenderer.Properties.cs.

7.111.5.45 SpaceAfterLine

```
double VectSharp.Markdown.MarkdownRenderer.SpaceAfterLine = 0.25  [get], [set]
```

The space after each line of text. This value will be multiplied by the BaseFontSize.

Definition at line 160 of file MarkdownRenderer.Properties.cs.

7.111.5.46 SpaceAfterParagraph

```
double VectSharp.Markdown.MarkdownRenderer.SpaceAfterParagraph = 0.75  [get], [set]
```

The space after each text paragraph. This value will be multiplied by the BaseFontSize.

Definition at line 155 of file MarkdownRenderer.Properties.cs.

7.111.5.47 SpaceBeforeHeading

```
double VectSharp.Markdown.MarkdownRenderer.SpaceBeforeHeading = 0.25  [get], [set]
```

The space before each heading. This value will be multiplied by the font size of the heading.

Definition at line 165 of file MarkdownRenderer.Properties.cs.

7.111.5.48 SpaceBeforeParagaph

```
double VectSharp.Markdown.MarkdownRenderer.SpaceBeforeParagaph = 0  [get], [set]
```

The space before each text paragraph. This value will be multiplied by the BaseFontSize.

Definition at line 150 of file MarkdownRenderer.Properties.cs.

7.111.5.49 SubscriptShift

```
double VectSharp.Markdown.MarkdownRenderer.SubscriptShift = 0.14  [get], [set]
```

The downwards shift in the baseline for subscript elements. This value will be multiplied by the current font size.

Definition at line 205 of file MarkdownRenderer.Properties.cs.

7.111.5.50 SubSuperscriptFontSize

```
double VectSharp.Markdown.MarkdownRenderer.SubSuperscriptFontSize = 0.7  [get], [set]
```

The font size for subscripts and superscripts. This value will be multiplied by the current font size.

Definition at line 195 of file MarkdownRenderer.Properties.cs.

7.111.5.51 SuperscriptShift

```
double VectSharp.Markdown.MarkdownRenderer.SuperscriptShift = 0.33  [get], [set]
```

The upwards shift in the baseline for superscript elements. This value will be multiplied by the current font size.

Definition at line 200 of file MarkdownRenderer.Properties.cs.

7.111.5.52 SyntaxHighlighter

```
Func<string, string, List<List<FormattedString> > > VectSharp.Markdown.MarkdownRenderer.↩
SyntaxHighlighter = VectSharp.Markdown.SyntaxHighlighter.GetSyntaxHighlightedLines  [get],
[set]
```

A method used for syntax highlighting. The first argument should be the source code to highlight, while the second parameter is the name of the language to use for the highlight. The method should return a list of lists of FormattedStrings, with each list of FormattedStrings representing a line. For each code block, if the method returns `null`, no syntax highlighting is used.

Definition at line 260 of file MarkdownRenderer.Properties.cs.

7.111.5.53 TableCellMargins

```
Margins VectSharp.Markdown.MarkdownRenderer.TableCellMargins = new Margins(5, 0, 5, 0)  [get],
[set]
```

The margins for table cells.

Definition at line 114 of file MarkdownRenderer.Properties.cs.

7.111.5.54 TableHeaderRowSeparatorColour

```
Colour VectSharp.Markdown.MarkdownRenderer.TableHeaderRowSeparatorColour = Colours.Black  [get], [set]
```

The colour for the line separating the table header row from normal rows.

Definition at line 375 of file MarkdownRenderer.Properties.cs.

7.111.5.55 TableHeaderRowSeparatorThickness

```
double VectSharp.Markdown.MarkdownRenderer.TableHeaderRowSeparatorThickness = 2  [get], [set]
```

The thickness of the line separating the table header row from normal rows.

Definition at line 385 of file MarkdownRenderer.Properties.cs.

7.111.5.56 TableHeaderSeparatorThickness

```
double VectSharp.Markdown.MarkdownRenderer.TableHeaderSeparatorThickness = 1  [get], [set]
```

The thickness of lines separating table rows from each other.

Definition at line 390 of file MarkdownRenderer.Properties.cs.

7.111.5.57 TableRowSeparatorColour

```
Colour VectSharp.Markdown.MarkdownRenderer.TableRowSeparatorColour = Colour.FromRgb(180, 180, 180)  [get], [set]
```

The colour for lines separating table rows from each other.

Definition at line 380 of file MarkdownRenderer.Properties.cs.

7.111.5.58 TableVAlign

```
VerticalAlignment VectSharp.Markdown.MarkdownRenderer.TableVAlign = VerticalAlignment.Middle  [get], [set]
```

The vertical alignment of table cells.

Definition at line 140 of file MarkdownRenderer.Properties.cs.

7.111.5.59 TaskListCheckedBullet

```
Graphics VectSharp.Markdown.MarkdownRenderer.TaskListCheckedBullet  [get], [set]
```

Initial value:

```
= new Func<Graphics>(() =>
        {
            Graphics tbr = new Graphics();
            GraphicsPath checkboxPath = new GraphicsPath().MoveTo(-0.7, -0.4).LineTo(-0.3, -0.4).Arc(-0.3,
      -0.2, 0.2, 3 * Math.PI / 2, 2 * Math.PI).LineTo(-0.1, 0.2).Arc(-0.3, 0.2, 0.2, 0, Math.PI /
      2).LineTo(-0.7, 0.4).Arc(-0.7, 0.2, 0.2, Math.PI / 2, Math.PI).LineTo(-0.9, -0.2).Arc(-0.7, -0.2, 0.2,
      Math.PI, 3 * Math.PI / 2).Close();
            tbr.FillPath(checkboxPath, Colour.FromRgb(240, 246, 249));
            tbr.StrokePath(checkboxPath, Colour.FromRgb(0, 114, 178), 0.075);
            GraphicsPath tickpath = new GraphicsPath().MoveTo(-0.75, -0.1).LineTo(-0.5, 0.15).LineTo(-0.1,
      -0.4);
            tbr.StrokePath(new GraphicsPath().MoveTo(-0.5, 0.15).LineTo(-0.1, -0.4), Colour.FromRgb(240,
      246, 249), 0.3, LineCaps.Round);
            tbr.StrokePath(tickpath, Colour.FromRgb(0, 158, 115), 0.2, LineCaps.Round);
            return tbr;
        })()
```

The bullet used for checked task list items.

Definition at line 301 of file MarkdownRenderer.Properties.cs.

7.111.5.60 TaskListUncheckedBullet

```
Graphics VectSharp.Markdown.MarkdownRenderer.TaskListUncheckedBullet  [get], [set]
```

Initial value:

```
= new Func<Graphics>(() =>
        {
            Graphics tbr = new Graphics();
            GraphicsPath checkboxPath = new GraphicsPath().MoveTo(-0.7, -0.4).LineTo(-0.3, -0.4).Arc(-0.3,
      -0.2, 0.2, 3 * Math.PI / 2, 2 * Math.PI).LineTo(-0.1, 0.2).Arc(-0.3, 0.2, 0.2, 0, Math.PI /
      2).LineTo(-0.7, 0.4).Arc(-0.7, 0.2, 0.2, Math.PI / 2, Math.PI).LineTo(-0.9, -0.2).Arc(-0.7, -0.2, 0.2,
      Math.PI, 3 * Math.PI / 2).Close();
            tbr.FillPath(checkboxPath, Colour.FromRgb(240, 246, 249));
            tbr.StrokePath(checkboxPath, Colour.FromRgb(0, 114, 178), 0.075);
            return tbr;
        })()
```

The bullet used for unchecked task list items.

Definition at line 287 of file MarkdownRenderer.Properties.cs.

7.111.5.61 ThematicBreakLineColour

```
Colour VectSharp.Markdown.MarkdownRenderer.ThematicBreakLineColour = Colour.FromRgb(180, 180,
200)  [get], [set]
```

The colour for thematic break lines.

Definition at line 335 of file MarkdownRenderer.Properties.cs.

7.111.5.62 ThematicBreakThickness

```
double VectSharp.Markdown.MarkdownRenderer.ThematicBreakThickness = 2  [get], [set]
```

The thickness of thematic break lines.

Definition at line 54 of file MarkdownRenderer.Properties.cs.

7.111.5.63 UnderlineThickness

```
double VectSharp.Markdown.MarkdownRenderer.UnderlineThickness = 0.075  [get], [set]
```

The thickness of underlines. This value will be multiplied by the font size of the element being underlined.

Definition at line 99 of file MarkdownRenderer.Properties.cs.

7.111.6 Event Documentation

7.111.6.1 BlockRendered

```
EventHandler<BlockRenderedEventArgs> VectSharp.Markdown.MarkdownRenderer.BlockRendered
```

Invoked just after rendering a Block.

Definition at line 44 of file MarkdownRenderer.Events.cs.

7.111.6.2 BlockRendering

```
EventHandler<BlockRenderingEventArgs> VectSharp.Markdown.MarkdownRenderer.BlockRendering
```

Invoked just before rendering a Block.

Definition at line 39 of file MarkdownRenderer.Events.cs.

7.111.6.3 InlineRendered

```
EventHandler<InlineRenderedEventArgs> VectSharp.Markdown.MarkdownRenderer.InlineRendered
```

Invoked just after rendering an Inline.

Definition at line 54 of file MarkdownRenderer.Events.cs.

7.111.6.4 InlineRendering

```
EventHandler<InlineRenderingEventArgs> VectSharp.Markdown.MarkdownRenderer.InlineRendering
```

Invoked just before rendering an Inline.

Definition at line 49 of file MarkdownRenderer.Events.cs.

7.111.6.5 LineMeasured

```
EventHandler<LineMeasuredEventArgs> VectSharp.Markdown.MarkdownRenderer.LineMeasured
```

Invoked after content in a Line has been measured, but before deciding whether to render the line on the current page or on a new page.

Definition at line 59 of file MarkdownRenderer.Events.cs.

7.111.6.6 LineRendered

```
EventHandler<LineEventArgs> VectSharp.Markdown.MarkdownRenderer.LineRendered
```

Invoked just after rendering a Line.

Definition at line 69 of file MarkdownRenderer.Events.cs.

7.111.6.7 LineRendering

```
EventHandler<LineEventArgs> VectSharp.Markdown.MarkdownRenderer.LineRendering
```

Invoked just before rendering a Line, after a new page has been created if necessary.

Definition at line 64 of file MarkdownRenderer.Events.cs.

7.111.6.8 PageFinished

```
EventHandler<PageFinishedEventArgs> VectSharp.Markdown.MarkdownRenderer.PageFinished
```

Invoked when a page is finished, before starting a new page (or at the end of the document).

Definition at line 34 of file MarkdownRenderer.Events.cs.

7.111.6.9 PageStarted

```
EventHandler<PageStartedEventArgs> VectSharp.Markdown.MarkdownRenderer.PageStarted
```

Invoked when a new page is started.

Definition at line 29 of file MarkdownRenderer.Events.cs.

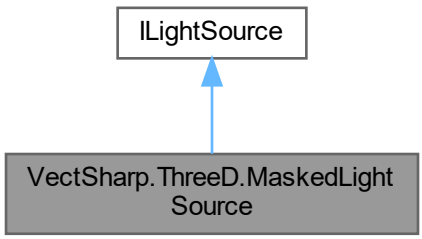
The documentation for this class was generated from the following files:

- VectSharp.Markdown/MarkdownRenderer.Blocks.cs
- VectSharp.Markdown/MarkdownRenderer.cs
- VectSharp.Markdown/MarkdownRenderer.Events.cs
- VectSharp.Markdown/MarkdownRenderer.Properties.cs
- VectSharp.Markdown/MarkdownRenderer.Inlines.cs
- VectSharp.Markdown/MarkdownRenderer.Tables.cs

7.112 VectSharp.ThreeD.MaskedLightSource Class Reference

Represents a point light source with a stencil in front of it.

Inheritance diagram for VectSharp.ThreeD.MaskedLightSource:



Public Member Functions

- MaskedLightSource (double intensity, Point3D position, NormalizedVector3D direction, double distance, GraphicsPath mask, double maskOrientation, double triangulationResolution)

  Creates a new MaskedLightSource by triangulating the specified GraphicsPath.
- MaskedLightSource (double intensity, Point3D position, NormalizedVector3D direction, double distance, IEnumerable< GraphicsPath > triangulatedMask, double maskOrientation)

  Creates a new MaskedLightSource using the specified triangulatedMask .
- LightIntensity GetLightAt (Point3D point)

  Computes the light intensity at the specified point, without taking into account any obstructions.

Parameters

point	*The Point3DElement at which the light intensity should be computed.*

Returns

- double GetObstruction (Point3D point, IEnumerable< Triangle3DElement > shadowingTriangles)

  Determines the amount of obstruction of the light that results at point due to the specified shadowingTriangles .

Parameters

point	*The Point3D at which the obstruction should be computed.*
shadowingTriangles	*A collection of Triangle3DElement casting shadows.*

Returns

1 if the light is completely obstructed, 0 if the light is completely visible, a value between these if the light is partially obstructed.

Properties

- bool CastsShadow = true `[get, set]`

  Determines whether the light casts a shadow or not.
- Point3D Position `[get]`

  The position of the light source.
- Point3D Origin `[get]`

  The projection of the Position on the mask plane along the light's Direction.
- NormalizedVector3D Direction `[get]`

  The direction of the light.
- double Distance `[get]`

  The distance between the light source and the mask plane.
- double Intensity `[get, set]`

  The base intensity of the light.
- double DistanceAttenuationExponent = 2 `[get, set]`

  An exponent determining how fast the light attenuates with increasing distance. Set to 0 to disable distance attenuation.
- double AngleAttenuationExponent = 1 `[get, set]`

  An exponent determining how fast the light attenuates away from the light's axis. Set to 0 to disable angular attenuation.

7.112.1 Detailed Description

Represents a point light source with a stencil in front of it.

Definition at line 385 of file Lights.cs.

7.112.2 Constructor & Destructor Documentation

7.112.2.1 MaskedLightSource() [1/2]

```
VectSharp.ThreeD.MaskedLightSource.MaskedLightSource (
            double intensity,
            Point3D position,
            NormalizedVector3D direction,
            double distance,
            GraphicsPath mask,
            double maskOrientation,
            double triangulationResolution )
```

Creates a new MaskedLightSource by triangulating the specified GraphicsPath.

Parameters

intensity	The base intensity of the light.
position	The position of the light source.
direction	The direction of the light.
distance	The distance between the light source and the mask plane.
mask	A GraphicsPath representing the transparent part of the mask.
maskOrientation	An angle in radians determining the orientation of the 2D mask in the mask plane.
triangulationResolution	The resolution to use to triangulate the *mask* .

Definition at line 437 of file Lights.cs.

7.112.2.2 MaskedLightSource() [2/2]

```
VectSharp.ThreeD.MaskedLightSource.MaskedLightSource (
            double intensity,
            Point3D position,
            NormalizedVector3D direction,
            double distance,
            IEnumerable< GraphicsPath > triangulatedMask,
            double maskOrientation )
```

Creates a new MaskedLightSource using the specified *triangulatedMask* .

Parameters

intensity	The base intensity of the light.
position	The position of the light source.
direction	The direction of the light.
distance	The distance between the light source and the mask plane.
triangulatedMask	A collection of GraphicsPaths representing the transparent part of the mask. Each GraphicsPath should represent a single triangle.
maskOrientation	An angle in radians determining the orientation of the 2D mask in the mask plane.

Definition at line 451 of file Lights.cs.

7.112.3 Member Function Documentation

7.112.3.1 GetLightAt()

```
LightIntensity VectSharp.ThreeD.MaskedLightSource.GetLightAt (
            Point3D point )
```

Computes the light intensity at the specified point, without taking into account any obstructions.

Parameters

point	The Point3DElement at which the light intensity should be computed.

Returns

Implements VectSharp.ThreeD.ILightSource.

Definition at line 483 of file Lights.cs.

7.112.3.2 GetObstruction()

```
double VectSharp.ThreeD.MaskedLightSource.GetObstruction (
            Point3D point,
            IEnumerable< Triangle3DElement > shadowingTriangles )
```

Determines the amount of obstruction of the light that results at *point* due to the specified *shadowingTriangles* .

Parameters

point	The Point3D at which the obstruction should be computed.
shadowingTriangles	A collection of Triangle3DElement casting shadows.

Returns

1 if the light is completely obstructed, 0 if the light is completely visible, a value between these if the light is partially obstructed.

Implements VectSharp.ThreeD.ILightSource.

Definition at line 556 of file Lights.cs.

7.112.4 Property Documentation

7.112.4.1 AngleAttenuationExponent

```
double VectSharp.ThreeD.MaskedLightSource.AngleAttenuationExponent = 1  [get], [set]
```

An exponent determining how fast the light attenuates away from the light's axis. Set to 0 to disable angular attenuation.

Definition at line 425 of file Lights.cs.

7.112.4.2 CastsShadow

```
bool VectSharp.ThreeD.MaskedLightSource.CastsShadow = true  [get], [set]
```

Determines whether the light casts a shadow or not.

Implements VectSharp.ThreeD.ILightSource.

Definition at line 388 of file Lights.cs.

7.112.4.3 Direction

```
NormalizedVector3D VectSharp.ThreeD.MaskedLightSource.Direction  [get]
```

The direction of the light.

Definition at line 403 of file Lights.cs.

7.112.4.4 Distance

```
double VectSharp.ThreeD.MaskedLightSource.Distance  [get]
```

The distance between the light source and the mask plane.

Definition at line 408 of file Lights.cs.

7.112.4.5 DistanceAttenuationExponent

```
double VectSharp.ThreeD.MaskedLightSource.DistanceAttenuationExponent = 2  [get], [set]
```

An exponent determining how fast the light attenuates with increasing distance. Set to 0 to disable distance attenuation.

Definition at line 420 of file Lights.cs.

7.112.4.6 Intensity

```
double VectSharp.ThreeD.MaskedLightSource.Intensity  [get], [set]
```

The base intensity of the light.

Definition at line 415 of file Lights.cs.

7.112.4.7 Origin

```
Point3D VectSharp.ThreeD.MaskedLightSource.Origin  [get]
```

The projection of the Position on the mask plane along the light's Direction.

Definition at line 398 of file Lights.cs.

7.112.4.8 Position

```
Point3D VectSharp.ThreeD.MaskedLightSource.Position  [get]
```

The position of the light source.

Definition at line 393 of file Lights.cs.

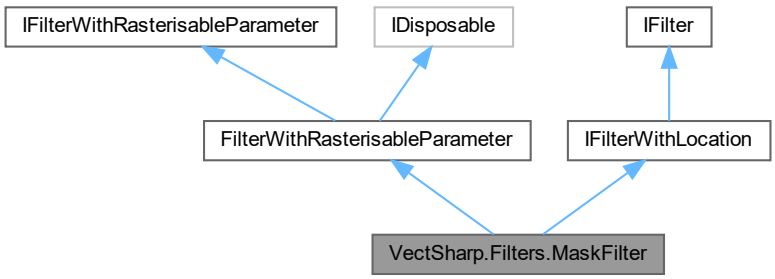
The documentation for this class was generated from the following file:

- VectSharp.ThreeD/Lights.cs

7.113 VectSharp.Filters.MaskFilter Class Reference

Represents a filter that uses the luminance of an image to mask another image.

Inheritance diagram for VectSharp.Filters.MaskFilter:



Public Member Functions

- MaskFilter (Graphics mask)

  Creates a new MaskFilter with the specified mask image.
- RasterImage Filter (RasterImage image, Rectangle bounds, double scale)

  Applies the filter to a RasterImage.

Parameters

image	*The RasterImage to which the filter will be applied.*
bounds	*The region on the graphics surface where the image will be drawn.*
scale	*The scale of the image with respect to the filter.*

Returns

A new RasterImage containing the filtered image. The source image is left unaltered.

Properties

- Point TopLeftMargin `[get]`

  Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.
- Point BottomRightMargin `[get]`

  Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.
- Graphics Mask `[get]`

  The image that is used to mask the input image.

7.113.1 Detailed Description

Represents a filter that uses the luminance of an image to mask another image.

Definition at line 27 of file MaskFilter.cs.

7.113.2 Constructor & Destructor Documentation

7.113.2.1 MaskFilter()

```
VectSharp.Filters.MaskFilter.MaskFilter (
            Graphics mask )
```

Creates a new MaskFilter with the specified mask image.

Parameters

mask	The image that is used to mask the input image.

Definition at line 43 of file MaskFilter.cs.

7.113.3 Member Function Documentation

7.113.3.1 Filter()

```
RasterImage VectSharp.Filters.MaskFilter.Filter (
            RasterImage image,
            Rectangle bounds,
            double scale )
```

Applies the filter to a RasterImage.

Parameters

image	The RasterImage to which the filter will be applied.
bounds	The region on the graphics surface where the image will be drawn.
scale	The scale of the image with respect to the filter.

Returns

A new RasterImage containing the filtered image. The source *image* is left unaltered.

Implements VectSharp.Filters.IFilterWithLocation.

Definition at line 47 of file MaskFilter.cs.

7.113.4 Property Documentation

7.113.4.1 BottomRightMargin

```
Point VectSharp.Filters.MaskFilter.BottomRightMargin  [get]
```

Determines how much the area of the filter's subject should be expanded on the bottom-right to accommodate the results of the filter.

Implements VectSharp.Filters.IFilter.

Definition at line 32 of file MaskFilter.cs.

7.113.4.2 Mask

```
Graphics VectSharp.Filters.MaskFilter.Mask  [get]
```

The image that is used to mask the input image.

Definition at line 37 of file MaskFilter.cs.

7.113.4.3 TopLeftMargin

```
Point VectSharp.Filters.MaskFilter.TopLeftMargin [get]
```

Determines how much the area of the filter's subject should be expanded on the top-left to accommodate the results of the filter.

Implements VectSharp.Filters.IFilter.

Definition at line 30 of file MaskFilter.cs.

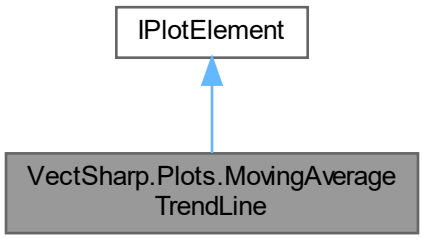
The documentation for this class was generated from the following file:

- VectSharp/Filters/MaskFilter.cs

7.114 VectSharp.Plots.MovingAverageTrendLine Class Reference

A plot element that draws a moving average trendline.

Inheritance diagram for VectSharp.Plots.MovingAverageTrendLine:



Public Member Functions

- MovingAverageTrendLine (IReadOnlyList< IReadOnlyList< double > > data, int period, ICoordinateSystem< IReadOnlyList< double > > coordinateSystem)

  Create a new MovingAverageTrendLine instance from the specified data.
- MovingAverageTrendLine (IReadOnlyList<(double, double)> data, int period, IContinuousCoordinateSystem coordinateSystem)

  Create a new MovingAverageTrendLine instance from the specified data.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IReadOnlyList< IReadOnlyList< double > > Data `[get, set]`

  The data used to compute the moving average. These must be sorted in a sensible way.
- Func< int, double[], double > Weight = (int i, double[] d) => 1 `[get, set]`

  The weight function. This should accept two parameters: an `int` representing the difference in index between the point being weighted and the "focus point", and a `double[]` representing the difference in coordinates between the two points. It should return a `double` representing the weight (it does not need to be normalised).
- int Period `[get, set]`

  The number of points that are averaged to obtain the value for each point. This must be ≥ 0.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180, 180, 180) } `[get, set]`

  Presentation attributes for the trendline.
- string Tag `[get, set]`

  A tag to identify the trendline in the plot.
- ICoordinateSystem< IReadOnlyList< double > > CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.

7.114.1 Detailed Description

A plot element that draws a moving average trendline.

Definition at line 1074 of file Trendlines.cs.

7.114.2 Constructor & Destructor Documentation

7.114.2.1 MovingAverageTrendLine() [1/2]

```
VectSharp.Plots.MovingAverageTrendLine.MovingAverageTrendLine (
            IReadOnlyList< IReadOnlyList< double > > data,
            int period,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem )
```

Create a new MovingAverageTrendLine instance from the specified data.

Parameters

data	The data used to compute the moving average. These must be sorted in a sensible way.
period	The number of points that are averaged to obtain the value for each point. This must be ≥ 0.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Exceptions

ArgumentOutOfRangeException	Thrown if the *period* is < 0.

Definition at line 1117 of file Trendlines.cs.

7.114.2.2 MovingAverageTrendLine() [2/2]

```
VectSharp.Plots.MovingAverageTrendLine.MovingAverageTrendLine (
            IReadOnlyList<(double, double)> data,
            int period,
            IContinuousCoordinateSystem coordinateSystem )
```

Create a new MovingAverageTrendLine instance from the specified data.

Parameters

data	The data used to compute the moving average. These must be sorted in a sensible way.
period	The number of points that are averaged to obtain the value for each point. This must be ≥ 0.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Exceptions

ArgumentOutOfRangeException	Thrown if the *period* is < 0.

Definition at line 1136 of file Trendlines.cs.

7.114.3 Member Function Documentation

7.114.3.1 Plot()

```
void VectSharp.Plots.MovingAverageTrendLine.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 1139 of file Trendlines.cs.

7.114.4 Property Documentation

7.114.4.1 CoordinateSystem

```
ICoordinateSystem<IReadOnlyList<double> > VectSharp.Plots.MovingAverageTrendLine.Coordinate↩
System [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 1107 of file Trendlines.cs.

7.114.4.2 Data

```
IReadOnlyList<IReadOnlyList<double> > VectSharp.Plots.MovingAverageTrendLine.Data [get],
[set]
```

The data used to compute the moving average. These must be sorted in a sensible way.

Definition at line 1079 of file Trendlines.cs.

7.114.4.3 Period

```
int VectSharp.Plots.MovingAverageTrendLine.Period [get], [set]
```

The number of points that are averaged to obtain the value for each point. This must be ≥ 0.

Definition at line 1092 of file Trendlines.cs.

7.114.4.4 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.MovingAverageTrendLine.PresentationAttributes
= new PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180,
180, 180) } [get], [set]
```

Presentation attributes for the trendline.

Definition at line 1097 of file Trendlines.cs.

7.114.4.5 Tag

```
string VectSharp.Plots.MovingAverageTrendLine.Tag [get], [set]
```

A tag to identify the trendline in the plot.

Definition at line 1102 of file Trendlines.cs.

7.114.4.6 Weight

```
Func<int, double[], double> VectSharp.Plots.MovingAverageTrendLine.Weight = (int i, double[]
d) => 1  [get], [set]
```

The weight function. This should accept two parameters: an `int` representing the difference in index between the point being weighted and the "focus point", and a `double`[] representing the difference in coordinates between the two points. It should return a `double` representing the weight (it does not need to be normalised).

Definition at line 1087 of file Trendlines.cs.

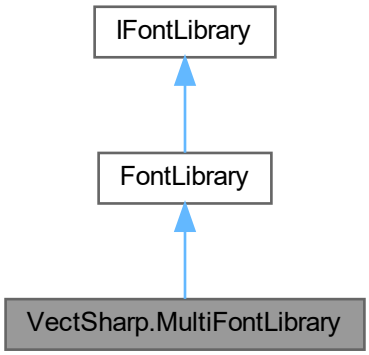
The documentation for this class was generated from the following file:

- VectSharp.Plots/Trendlines.cs

7.115 VectSharp.MultiFontLibrary Class Reference

A font library that tries to resolve fonts using other font libraries.

Inheritance diagram for VectSharp.MultiFontLibrary:



Public Member Functions

- MultiFontLibrary (params FontLibrary[ ] libraries)

  Creates a new MultiFontLibrary resolving fonts using the specified libraries .
- MultiFontLibrary (IEnumerable< FontLibrary > libraries)

  Creates a new MultiFontLibrary resolving fonts using the specified libraries .
- override FontFamily ResolveFontFamily (string fontFamily)

  Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised.

Parameters

fontFamily	*The name of the font family to create, or the path to a TTF file.*

Returns

If the font family name or the true type file is valid, a FontFamily object corresponding to the specified font family.

- override FontFamily ResolveFontFamily (FontFamily.StandardFontFamilies standardFontFamily)

  Create a new font family from the specified standard font family name.

Parameters

standardFontFamily	*The standard name of the font family.*

Returns

A FontFamily object corresponding to the specified font family.

7.115.1 Detailed Description

A font library that tries to resolve fonts using other font libraries.

Definition at line 654 of file FontLibrary.cs.

7.115.2 Constructor & Destructor Documentation

7.115.2.1 MultiFontLibrary() [1/2]

```
VectSharp.MultiFontLibrary.MultiFontLibrary (
            params FontLibrary[] libraries )
```

Creates a new MultiFontLibrary resolving fonts using the specified *libraries* .

Parameters

libraries	The font libraries that will be used, in order, to resolve the font families.

Exceptions

ArgumentException	Thrown if the *libraries* do not contain any element.

Definition at line 664 of file FontLibrary.cs.

7.115.2.2 MultiFontLibrary() [2/2]

```
VectSharp.MultiFontLibrary.MultiFontLibrary (
            IEnumerable< FontLibrary > libraries )
```

Creates a new MultiFontLibrary resolving fonts using the specified *libraries* .

Parameters

libraries	The font libraries that will be used, in order, to resolve the font families.

Exceptions

ArgumentException	Thrown if the *libraries* do not contain any element.

Definition at line 672 of file FontLibrary.cs.

7.115.3 Member Function Documentation

7.115.3.1 ResolveFontFamily() [1/2]

```
override FontFamily VectSharp.MultiFontLibrary.ResolveFontFamily (
            FontFamily.StandardFontFamilies standardFontFamily )  [virtual]
```

Create a new font family from the specified standard font family name.

Parameters

standardFontFamily	The standard name of the font family.

Returns

A FontFamily object corresponding to the specified font family.

Implements VectSharp.FontLibrary.

Definition at line 699 of file FontLibrary.cs.

7.115.3.2 ResolveFontFamily() [2/2]

```
override FontFamily VectSharp.MultiFontLibrary.ResolveFontFamily (
            string fontFamily )  [virtual]
```

Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised.

Parameters

fontFamily	The name of the font family to create, or the path to a TTF file.

Returns

If the font family name or the true type file is valid, a FontFamily object corresponding to the specified font family.

Implements VectSharp.FontLibrary.

Definition at line 683 of file FontLibrary.cs.

The documentation for this class was generated from the following file:

- VectSharp/FontLibrary.cs

7.116 VectSharp.Fonts.Nimbus Class Reference

Contains an IFontLibrary providing access to the Nimbus family of standard fonts (used e.g. by MuPDF).

Properties

- static IFontLibrary Library `[get]`
  The font library.

7.116.1 Detailed Description

Contains an IFontLibrary providing access to the Nimbus family of standard fonts (used e.g. by MuPDF).

Definition at line 21 of file Nimbus.cs.

7.116.2 Property Documentation

7.116.2.1 Library

```
IFontLibrary VectSharp.Fonts.Nimbus.Library  [static], [get]
```

The font library.

Definition at line 26 of file Nimbus.cs.

The documentation for this class was generated from the following file:

- VectSharp.Fonts.Nimbus/Nimbus.cs

7.117 VectSharp.ThreeD.ObjectFactory Class Reference

A static class containing methods to create complex 3D objects.

Static Public Member Functions

- static List< Element3D > CreateCube (Point3D center, double size, IEnumerable< IMaterial > fill, string tag=null, int zIndex=0)

  Creates a cube.
- static List< Element3D > CreateCuboid (Point3D center, double sizeX, double sizeY, double sizeZ, IEnumerable< IMaterial > fill, string tag=null, int zIndex=0)

  Creates a cuboid.
- static List< Element3D > CreateRectangle (Point3D point1, Point3D point2, Point3D point3, Point3D point4, IEnumerable< IMaterial > fill, string tag=null, int zIndex=0)

  Creates a quadrilater. All the vertices need not be coplanar.
- static List< Element3D > CreateRectangle (Point3D point1, Point3D point2, Point3D point3, Point3D point4, NormalizedVector3D point1Normal, NormalizedVector3D point2Normal, NormalizedVector3D point3Normal, NormalizedVector3D point4Normal, IEnumerable< IMaterial > fill, string tag=null, int zIndex=0)

  Creates a quadrilater, specifying the vertex normals at the four vertices. All the vertices need not be coplanar.
- static List< Element3D > CreateSphere (Point3D center, double radius, int steps, IEnumerable< IMaterial > fill, string tag=null, int zIndex=0)

  Creates a sphere.
- static List< Element3D > CreateTetrahedron (Point3D center, double radius, IEnumerable< IMaterial > fill, string tag=null, int zIndex=0)

  Creates a tetrahedron inscribed in a sphere.
- static List< Element3D > CreatePolygon (GraphicsPath polygon2D, double triangulationResolution, Point3D origin, NormalizedVector3D xAxis, NormalizedVector3D yAxis, bool reverseTriangles, IEnumerable< IMaterial > fill, string tag=null, int zIndex=0)

  Creates a flat polygon.
- static List< Element3D > CreatePrism (GraphicsPath polygonBase2D, double triangulationResolution, Point3D bottomOrigin, Point3D topOrigin, NormalizedVector3D baseXAxis, NormalizedVector3D baseYAxis, IEnumerable< IMaterial > fill, string tag=null, int zIndex=0)

  Creates a prism with the specified base.
- static List< Element3D > CreateWireframe (IEnumerable< Element3D > object3D, Colour colour, double thickness=1, LineCaps lineCap=LineCaps.Butt, LineDash? lineDash=null, string tag=null, int zIndex=0)

  Creates a wireframe from a collection of Element3Ds.
- static List< Element3D > CreatePoints (IEnumerable< Element3D > object3D, Colour colour, double diameter=1, string tag=null, int zIndex=0)

  Obtains a list of Point3DElement corresponding to the vertices of a list of Element3Ds.

7.117.1 Detailed Description

A static class containing methods to create complex 3D objects.

Definition at line 28 of file ObjectFactory.cs.

7.117.2 Member Function Documentation

7.117.2.1 CreateCube()

```
static List< Element3D > VectSharp.ThreeD.ObjectFactory.CreateCube (
            Point3D center,
            double size,
            IEnumerable< IMaterial > fill,
            string tag = null,
            int zIndex = 0 )  [static]
```

Creates a cube.

Parameters

center	The centre of the cube.
size	The length of each side of the cube.
fill	A collection of materials that will be applied to the Triangle3DElements returned by this method.
tag	A tag that will be applied to the Triangle3DElements returned by this method.
zIndex	A z-index that will be applied to the Triangle3DElements returned by this method.

Returns

A list of Triangle3DElements that constitute the cube.

Definition at line 39 of file ObjectFactory.cs.

7.117.2.2 CreateCuboid()

```
static List< Element3D > VectSharp.ThreeD.ObjectFactory.CreateCuboid (
            Point3D center,
            double sizeX,
            double sizeY,
            double sizeZ,
            IEnumerable< IMaterial > fill,
            string tag = null,
            int zIndex = 0 )  [static]
```

Creates a cuboid.

Parameters

center	The centre of the cube.
sizeX	The length of the sides of the cube parallel to the x axis.
sizeY	The length of the sides of the cube parallel to the y axis.
sizeZ	The length of the sides of the cube parallel to the z axis.
fill	A collection of materials that will be applied to the Triangle3DElements returned by this method.
tag	A tag that will be applied to the Triangle3DElements returned by this method.
zIndex	A z-index that will be applied to the Triangle3DElements returned by this method.

Returns

A list of Triangle3DElements that constitute the cuboid.

Definition at line 55 of file ObjectFactory.cs.

7.117.2.3 CreatePoints()

```
static List< Element3D > VectSharp.ThreeD.ObjectFactory.CreatePoints (
            IEnumerable< Element3D > object3D,
            Colour colour,
            double diameter = 1,
            string tag = null,
            int zIndex = 0 )  [static]
```

Obtains a list of Point3DElement corresponding to the vertices of a list of Element3Ds.

Parameters

object3D	The collection of Element3Ds. Point3DElements are ignored.
colour	The colour of the Point3DElements returned by this method.
diameter	The diameter of the Point3DElements returned by this method.
tag	A tag that will be applied to the Point3DElements returned by this method.
zIndex	A z-index that will be applied to the Point3DElements returned by this method.

Returns

A list of Point3DElements corresponding to the vertices of the Element3Ds.

Definition at line 412 of file ObjectFactory.cs.

7.117.2.4 CreatePolygon()

```
static List< Element3D > VectSharp.ThreeD.ObjectFactory.CreatePolygon (
            GraphicsPath polygon2D,
            double triangulationResolution,
            Point3D origin,
            NormalizedVector3D xAxis,
            NormalizedVector3D yAxis,
            bool reverseTriangles,
            IEnumerable< IMaterial > fill,
            string tag = null,
            int zIndex = 0 )  [static]
```

Creates a flat polygon.

Parameters

polygon2D	A 2D GraphicsPath representing the polygon.
triangulationResolution	The resolution that will be used to linearise curve segments in the GraphicsPath.
origin	A Point3D that will correspond to the origin of the 2D reference system.
xAxis	A NormalizedVector3D that will correspond to the x axis of the 2D reference system. This will be orthonormalised to the *yAxis* .
yAxis	A NormalizedVector3D that will correspond to the y axis of the 2D reference system.
reverseTriangles	Indicates whether the order of the points (and thus the normals) of all the triangles returned by this method should be reversed.
fill	A collection of materials that will be applied to the Triangle3DElements returned by this method.
tag	A tag that will be applied to the Triangle3DElements returned by this method.
zIndex	A z-index that will be applied to the Triangle3DElements returned by this method.

Returns

A list of Triangle3DElements that constitute the polygon.

Definition at line 273 of file ObjectFactory.cs.

7.117.2.5 CreatePrism()

```
static List< Element3D > VectSharp.ThreeD.ObjectFactory.CreatePrism (
            GraphicsPath polygonBase2D,
            double triangulationResolution,
            Point3D bottomOrigin,
            Point3D topOrigin,
            NormalizedVector3D baseXAxis,
            NormalizedVector3D baseYAxis,
            IEnumerable< IMaterial > fill,
            string tag = null,
            int zIndex = 0 )  [static]
```

Creates a prism with the specified base.

Parameters

polygonBase2D	A 2D GraphicsPath representing the base of the prism.
triangulationResolution	The resolution that will be used to linearise curve segments in the GraphicsPath.
bottomOrigin	A Point3D that will correspond to the origin of the 2D reference system of the bottom base.
topOrigin	A Point3D that will correspond to the origin of the 2D reference system of the top base.
baseXAxis	A NormalizedVector3D that will correspond to the x axis of the 2D reference system of the bases. This will be orthonormalised to the *baseYAxis* .
baseYAxis	A NormalizedVector3D that will correspond to the y axis of the 2D reference system of the bases.
fill	A collection of materials that will be applied to the Triangle3DElements returned by this method.
tag	A tag that will be applied to the Triangle3DElements returned by this method.
zIndex	A z-index that will be applied to the Triangle3DElements returned by this method.

Returns

A list of Triangle3DElements that constitute the prism.

Definition at line 314 of file ObjectFactory.cs.

7.117.2.6 CreateRectangle() [1/2]

```
static List< Element3D > VectSharp.ThreeD.ObjectFactory.CreateRectangle (
            Point3D point1,
            Point3D point2,
            Point3D point3,
            Point3D point4,
            IEnumerable< IMaterial > fill,
            string tag = null,
            int zIndex = 0 )  [static]
```

Creates a quadrilater. All the vertices need not be coplanar.

Parameters

point1	The first vertex of the quadrilater.
point2	The second vertex of the quadrilater.
point3	The third vertex of the quadrilater.
point4	The fourth vertex of the quadrilater.
fill	A collection of materials that will be applied to the Triangle3DElements returned by this method.
tag	A tag that will be applied to the Triangle3DElements returned by this method.
zIndex	A z-index that will be applied to the Triangle3DElements returned by this method.

Returns

A list containing two Triangle3DElements representing the quadrilater.

Definition at line 93 of file ObjectFactory.cs.

7.117.2.7 CreateRectangle() [2/2]

```
static List< Element3D > VectSharp.ThreeD.ObjectFactory.CreateRectangle (
            Point3D point1,
            Point3D point2,
            Point3D point3,
            Point3D point4,
            NormalizedVector3D point1Normal,
            NormalizedVector3D point2Normal,
            NormalizedVector3D point3Normal,
            NormalizedVector3D point4Normal,
            IEnumerable< IMaterial > fill,
            string tag = null,
            int zIndex = 0 )  [static]
```

Creates a quadrilater, specifying the vertex normals at the four vertices. All the vertices need not be coplanar.

Parameters

point1	The first vertex of the quadrilater.
point2	The second vertex of the quadrilater.
point3	The third vertex of the quadrilater.
point4	The fourth vertex of the quadrilater.
point1Normal	The vertex normal at the first vertex of the quadrilater.
point2Normal	The vertex normal at the second vertex of the quadrilater.
point3Normal	The vertex normal at the third vertex of the quadrilater.
point4Normal	The vertex normal at the fourth vertex of the quadrilater.
fill	A collection of materials that will be applied to the Triangle3DElements returned by this method.
tag	A tag that will be applied to the Triangle3DElements returned by this method.
zIndex	A z-index that will be applied to the Triangle3DElements returned by this method.

Returns

A list containing two Triangle3DElements representing the quadrilater.

Definition at line 123 of file ObjectFactory.cs.

7.117.2.8 CreateSphere()

```
static List< Element3D > VectSharp.ThreeD.ObjectFactory.CreateSphere (
            Point3D center,
            double radius,
            int steps,
            IEnumerable< IMaterial > fill,
            string tag = null,
            int zIndex = 0 )  [static]
```

Creates a sphere.

Parameters

center	The centre of the sphere.
radius	The radius of the sphere.
steps	The number of meridians and parallels to use when generating the sphere.
fill	A collection of materials that will be applied to the Triangle3DElements returned by this method.
tag	A tag that will be applied to the Triangle3DElements returned by this method.
zIndex	A z-index that will be applied to the Triangle3DElements returned by this method.

Returns

A list of Triangle3DElements that constitute the sphere.

Definition at line 148 of file ObjectFactory.cs.

7.117.2.9 CreateTetrahedron()

```
static List< Element3D > VectSharp.ThreeD.ObjectFactory.CreateTetrahedron (
            Point3D center,
            double radius,
            IEnumerable< IMaterial > fill,
            string tag = null,
            int zIndex = 0 )  [static]
```

Creates a tetrahedron inscribed in a sphere.

Parameters

center	The centre of the tetrahedron.
radius	The radius of the sphere in which the tetrahedron is inscribed.
fill	A collection of materials that will be applied to the Triangle3DElements returned by this method.
tag	A tag that will be applied to the Triangle3DElements returned by this method.
zIndex	A z-index that will be applied to the Triangle3DElements returned by this method.

Returns

A list of Triangle3DElements that constitute the sphere.

Definition at line 238 of file ObjectFactory.cs.

7.117.2.10 CreateWireframe()

```
static List< Element3D > VectSharp.ThreeD.ObjectFactory.CreateWireframe (
            IEnumerable< Element3D > object3D,
            Colour colour,
            double thickness = 1,
            LineCaps lineCap = LineCaps.Butt,
            LineDash?  lineDash = null,
            string tag = null,
            int zIndex = 0 )  [static]
```

Creates a wireframe from a collection of Element3Ds.

Parameters

object3D	The collection of Element3Ds. Line3DElements and Point3DElements are ignored.
colour	The colour of the Line3DElements returned by this method.
thickness	The thickness of the Line3DElements returned by this method.
lineCap	The line cap of the Line3DElements returned by this method.
lineDash	The line dash of the Line3DElements returned by this method.
tag	A tag that will be applied to the Line3DElements returned by this method.
zIndex	A z-index that will be applied to the Line3DElements returned by this method.

Returns

A list of Line3DElements that constitute the wireframe.

Definition at line 370 of file ObjectFactory.cs.

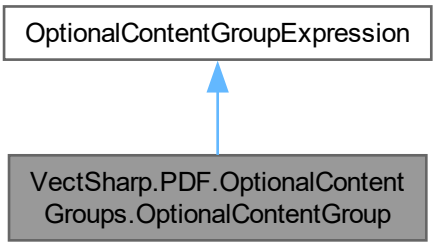
The documentation for this class was generated from the following file:

- VectSharp.ThreeD/ObjectFactory.cs

7.118 VectSharp.PDF.OptionalContentGroups.OptionalContentGroup Class Reference

Represents an optional content group.

Inheritance diagram for VectSharp.PDF.OptionalContentGroups.OptionalContentGroup:



Public Types

- enum Intents

  Possible intended uses of graphics in an OptionalContentGroup.

Public Member Functions

- OptionalContentGroup (string name, bool defaultState=true, bool? defaultViewState=null, bool? defaultPrintState=null, bool? defaultExportState=null, Intents intent=Intents.View)

  Create a new OptionalContentGroup.
- override string ToString ()

Static Public Member Functions

- static string CreateSetOCGLink (IReadOnlyList< OptionalContentGroup > on=null, IReadOnlyList< OptionalContentGroup > off=null, IReadOnlyList< OptionalContentGroup > toggle=null)

  Creates a link destination that sets the state of one or more OptionalContentGroups.

Properties

- override OptionalContentGroupBooleanOperation Operation `[get]`
- string Name `[get, set]`

  The name of the OptionalContentGroup.
- Intents Intent `[get, set]`

  The intended use of graphics in the OptionalContentGroup.
- bool DefaultState `[get, set]`

  The default state of the OptionalContentGroup when the document is opened.
- bool? DefaultViewState `[get, set]`

  The default state of the OptionalContentGroup when the document is opened for visualisation. Set this to `null` to use the DefaultState.
- bool? DefaultPrintState `[get, set]`

  The default state of the OptionalContentGroup when the document is opened for printing. Set this to `null` to use the DefaultState.
- bool? DefaultExportState `[get, set]`

  The default state of the OptionalContentGroup when the document is opened for exporting. Set this to `null` to use the DefaultState.

7.118.1 Detailed Description

Represents an optional content group.

Definition at line 157 of file OptionalContentGroups.cs.

7.118.2 Member Enumeration Documentation

7.118.2.1 Intents

```
enum VectSharp.PDF.OptionalContentGroups.OptionalContentGroup.Intents
```

Possible intended uses of graphics in an OptionalContentGroup.

Definition at line 170 of file OptionalContentGroups.cs.

7.118.3 Constructor & Destructor Documentation

7.118.3.1 OptionalContentGroup()

```
VectSharp.PDF.OptionalContentGroups.OptionalContentGroup.OptionalContentGroup (
            string name,
            bool defaultState = true,
            bool?  defaultViewState = null,
            bool?  defaultPrintState = null,
            bool?  defaultExportState = null,
            Intents intent = Intents.View )
```

Create a new OptionalContentGroup.

Parameters

name	The name of the OptionalContentGroup.
defaultState	The default state of the OptionalContentGroup when the document is opened.
defaultViewState	The default state of the OptionalContentGroup when the document is opened for visualisation. Set this to `null` to use the DefaultState.
defaultPrintState	The default state of the OptionalContentGroup when the document is opened for printing. Set this to `null` to use the DefaultState.
defaultExportState	The default state of the OptionalContentGroup when the document is opened for exporting. Set this to `null` to use the DefaultState.
intent	The intended use of graphics in the OptionalContentGroup.

Definition at line 222 of file OptionalContentGroups.cs.

7.118.4 Member Function Documentation

7.118.4.1 CreateSetOCGLink()

```
static string VectSharp.PDF.OptionalContentGroups.OptionalContentGroup.CreateSetOCGLink (
            IReadOnlyList< OptionalContentGroup > on = null,
            IReadOnlyList< OptionalContentGroup > off = null,
            IReadOnlyList< OptionalContentGroup > toggle = null )  [static]
```

Creates a link destination that sets the state of one or more OptionalContentGroups.

Parameters

on	OptionalContentGroups to turn on.
off	OptionalContentGroups to turn off.
toggle	OptionalContentGroups to toggle.

Returns

A string representation of an action that sets the specified optional content group states. Use this within the `linkDestionations` parameter of the PDFContextInterpreter.CreatePDFDocument method.

Exceptions

ArgumentException	Thrown if no OptionalContentGroups are provided in any of the arguments.

Definition at line 246 of file OptionalContentGroups.cs.

7.118.4.2 ToString()

```
override string VectSharp.PDF.OptionalContentGroups.OptionalContentGroup.ToString ( )
```

Definition at line 233 of file OptionalContentGroups.cs.

7.118.5 Property Documentation

7.118.5.1 DefaultExportState

```
bool?  VectSharp.PDF.OptionalContentGroups.OptionalContentGroup.DefaultExportState  [get],
[set]
```

The default state of the OptionalContentGroup when the document is opened for exporting. Set this to `null` to use the DefaultState.

Definition at line 206 of file OptionalContentGroups.cs.

7.118.5.2 DefaultPrintState

```
bool?  VectSharp.PDF.OptionalContentGroups.OptionalContentGroup.DefaultPrintState  [get], [set]
```

The default state of the OptionalContentGroup when the document is opened for printing. Set this to `null` to use the DefaultState.

Definition at line 201 of file OptionalContentGroups.cs.

7.118.5.3 DefaultState

```
bool VectSharp.PDF.OptionalContentGroups.OptionalContentGroup.DefaultState  [get], [set]
```

The default state of the OptionalContentGroup when the document is opened.

Definition at line 191 of file OptionalContentGroups.cs.

7.118.5.4 DefaultViewState

```
bool?  VectSharp.PDF.OptionalContentGroups.OptionalContentGroup.DefaultViewState  [get], [set]
```

The default state of the OptionalContentGroup when the document is opened for visualisation. Set this to `null` to use the DefaultState.

Definition at line 196 of file OptionalContentGroups.cs.

7.118.5.5 Intent

```
Intents VectSharp.PDF.OptionalContentGroups.OptionalContentGroup.Intent  [get], [set]
```

The intended use of graphics in the OptionalContentGroup.

Definition at line 186 of file OptionalContentGroups.cs.

7.118.5.6 Name

```
string VectSharp.PDF.OptionalContentGroups.OptionalContentGroup.Name  [get], [set]
```

The name of the OptionalContentGroup.

Definition at line 165 of file OptionalContentGroups.cs.

7.118.5.7 Operation

```
override OptionalContentGroupBooleanOperation VectSharp.PDF.OptionalContentGroups.Optional↩
ContentGroup.Operation  [get]
```

Definition at line 160 of file OptionalContentGroups.cs.

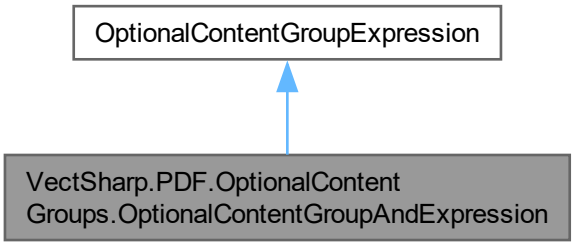
The documentation for this class was generated from the following file:

- VectSharp.PDF/OptionalContentGroups.cs

7.119 VectSharp.PDF.OptionalContentGroups.OptionalContentGroup↩ AndExpression Class Reference

An expression that is true when all its arguments are true.

Inheritance diagram for VectSharp.PDF.OptionalContentGroups.OptionalContentGroupAndExpression:



Public Member Functions

- OptionalContentGroupAndExpression (OptionalContentGroupExpression expression1, OptionalContentGroupExpression expression2, params OptionalContentGroupExpression[] expressions)

  Create a new OptionalContentGroupAndExpression.

Properties

- override OptionalContentGroupBooleanOperation Operation `[get]`

Additional Inherited Members

7.119.1 Detailed Description

An expression that is true when all its arguments are true.

Definition at line 324 of file OptionalContentGroups.cs.

7.119.2 Constructor & Destructor Documentation

7.119.2.1 OptionalContentGroupAndExpression()

```
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupAndExpression.OptionalContentGroup↩
AndExpression (
            OptionalContentGroupExpression expression1,
            OptionalContentGroupExpression expression2,
            params OptionalContentGroupExpression[] expressions )
```

Create a new OptionalContentGroupAndExpression.

Parameters

expression1	The first argument.
expression2	The second argument.
expressions	Additional arguments.

Definition at line 335 of file OptionalContentGroups.cs.

7.119.3 Property Documentation

7.119.3.1 Operation

```
override OptionalContentGroupBooleanOperation VectSharp.PDF.OptionalContentGroups.Optional↩
ContentGroupAndExpression.Operation  [get]
```

Definition at line 327 of file OptionalContentGroups.cs.

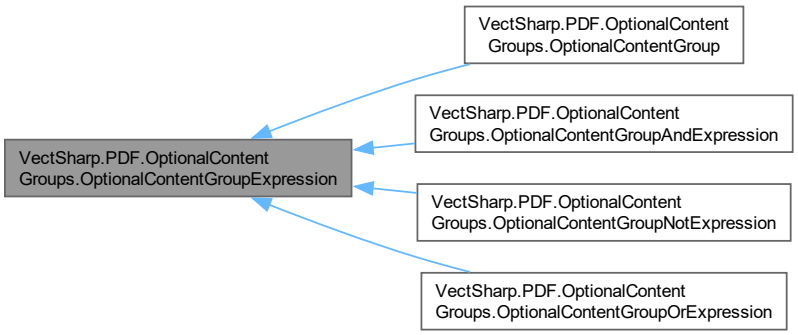
The documentation for this class was generated from the following file:

- VectSharp.PDF/OptionalContentGroups.cs

7.120 VectSharp.PDF.OptionalContentGroups.OptionalContentGroup↩ Expression Class Reference

Represents the result of a boolean operation on OptionalContentGroups.

Inheritance diagram for VectSharp.PDF.OptionalContentGroups.OptionalContentGroupExpression:



Public Member Functions

- override string ToString ()

Static Public Member Functions

- static OptionalContentGroupNotExpression operator! (OptionalContentGroupExpression ocge)

  Negates an OptionalContentGroupExpression.
- static bool operator false (OptionalContentGroupExpression _)

  Used to enable short circuit operators.
- static bool operator true (OptionalContentGroupExpression _)

  Used to enable short circuit operators.
- static OptionalContentGroupExpression operator& (OptionalContentGroupExpression ocge1, OptionalContentGroupExpression ocge2)

  Performs a conjunction of two OptionalContentGroupExpressions.
- static OptionalContentGroupExpression operator| (OptionalContentGroupExpression ocge1, OptionalContentGroupExpression ocge2)

  Performs a disjunction of two OptionalContentGroupExpressions.

Properties

- abstract OptionalContentGroupBooleanOperation Operation [get]

  The operation performed by the expression.
- virtual IReadOnlyList< OptionalContentGroupExpression > Arguments [get]

  The arguments of the expression.

7.120.1 Detailed Description

Represents the result of a boolean operation on OptionalContentGroups.

Definition at line 55 of file OptionalContentGroups.cs.

7.120.2 Member Function Documentation

7.120.2.1 operator false()

```
static bool VectSharp.PDF.OptionalContentGroups.OptionalContentGroupExpression.operator false
(
            OptionalContentGroupExpression _ )  [static]
```

Used to enable short circuit operators.

Parameters

↩ _↩	The expression.

Returns

Always `false`.

Definition at line 90 of file OptionalContentGroups.cs.

7.120.2.2 operator true()

```
static bool VectSharp.PDF.OptionalContentGroups.OptionalContentGroupExpression.operator true (
            OptionalContentGroupExpression _ )  [static]
```

Used to enable short circuit operators.

Parameters

↩ _↩	The expression.

Returns

Always `false`.

Definition at line 100 of file OptionalContentGroups.cs.

7.120.2.3 operator"!()

```
static OptionalContentGroupNotExpression VectSharp.PDF.OptionalContentGroups.OptionalContent↩
GroupExpression.operator!  (
            OptionalContentGroupExpression ocge )  [static]
```

Negates an OptionalContentGroupExpression.

Parameters

ocge	The OptionalContentGroupExpression to be negated.

Definition at line 80 of file OptionalContentGroups.cs.

7.120.2.4 operator&()

```
static OptionalContentGroupExpression VectSharp.PDF.OptionalContentGroups.OptionalContent↩
GroupExpression.operator& (
            OptionalContentGroupExpression ocge1,
            OptionalContentGroupExpression ocge2 )  [static]
```

Performs a conjunction of two OptionalContentGroupExpressions.

Parameters

ocge1	The first expression.
ocge2	The second expression.

Returns

An OptionalContentGroupExpression representing the conjunction of the two expressions.

Definition at line 111 of file OptionalContentGroups.cs.

7.120.2.5 operator"|()

```
static OptionalContentGroupExpression VectSharp.PDF.OptionalContentGroups.OptionalContent↩
GroupExpression.operator| (
            OptionalContentGroupExpression ocge1,
            OptionalContentGroupExpression ocge2 )  [static]
```

Performs a disjunction of two OptionalContentGroupExpressions.

Parameters

ocge1	The first expression.
ocge2	The second expression.

Returns

An OptionalContentGroupExpression representing the disjunction of the two expressions.

Definition at line 122 of file OptionalContentGroups.cs.

7.120.2.6 ToString()

```
override string VectSharp.PDF.OptionalContentGroups.OptionalContentGroupExpression.ToString (
)
```

Definition at line 130 of file OptionalContentGroups.cs.

7.120.3 Property Documentation

7.120.3.1 Arguments

```
virtual IReadOnlyList<OptionalContentGroupExpression> VectSharp.PDF.OptionalContentGroups.↩
OptionalContentGroupExpression.Arguments  [get]
```

The arguments of the expression.

Definition at line 65 of file OptionalContentGroups.cs.

7.120.3.2 Operation

```
abstract OptionalContentGroupBooleanOperation VectSharp.PDF.OptionalContentGroups.Optional↩
ContentGroupExpression.Operation [get]
```

The operation performed by the expression.

Definition at line 60 of file OptionalContentGroups.cs.

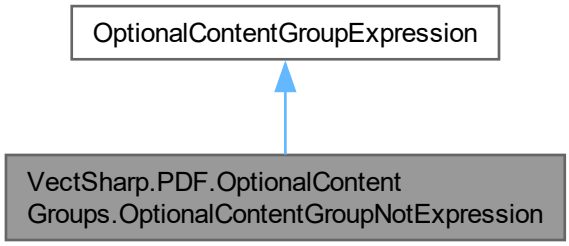
The documentation for this class was generated from the following file:

- VectSharp.PDF/OptionalContentGroups.cs

7.121 VectSharp.PDF.OptionalContentGroups.OptionalContentGroup↩ NotExpression Class Reference

An expression that is true when its argument is false.

Inheritance diagram for VectSharp.PDF.OptionalContentGroups.OptionalContentGroupNotExpression:



Public Member Functions

- OptionalContentGroupNotExpression (OptionalContentGroupExpression expression)
  Create a new OptionalContentGroupNotExpression.

Properties

- override OptionalContentGroupBooleanOperation Operation [get]

Additional Inherited Members

7.121.1 Detailed Description

An expression that is true when its argument is false.

Definition at line 309 of file OptionalContentGroups.cs.

7.121.2 Constructor & Destructor Documentation

7.121.2.1 OptionalContentGroupNotExpression()

```
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupNotExpression.OptionalContentGroup↩
NotExpression (
            OptionalContentGroupExpression expression )
```

Create a new OptionalContentGroupNotExpression.

Parameters

expression	The OptionalContentGroupNotExpression that should be negated.

Definition at line 318 of file OptionalContentGroups.cs.

7.121.3 Property Documentation

7.121.3.1 Operation

```
override OptionalContentGroupBooleanOperation VectSharp.PDF.OptionalContentGroups.Optional↩
ContentGroupNotExpression.Operation  [get]
```

Definition at line 312 of file OptionalContentGroups.cs.

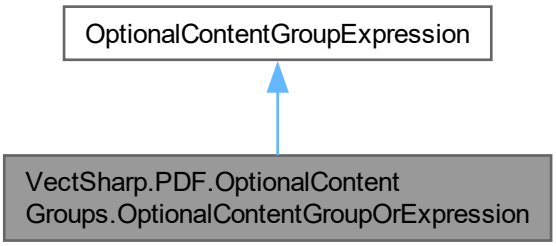
The documentation for this class was generated from the following file:

- VectSharp.PDF/OptionalContentGroups.cs

7.122 VectSharp.PDF.OptionalContentGroups.OptionalContentGroup↩ OrExpression Class Reference

An expression that is true when any of its arguments are true.

Inheritance diagram for VectSharp.PDF.OptionalContentGroups.OptionalContentGroupOrExpression:



Public Member Functions

- OptionalContentGroupOrExpression (OptionalContentGroupExpression expression1, OptionalContentGroupExpression expression2, params OptionalContentGroupExpression[] expressions)

  Create a new OptionalContentGroupOrExpression.

Properties

- override OptionalContentGroupBooleanOperation Operation `[get]`

Additional Inherited Members

7.122.1 Detailed Description

An expression that is true when any of its arguments are true.

Definition at line 341 of file OptionalContentGroups.cs.

7.122.2 Constructor & Destructor Documentation

7.122.2.1 OptionalContentGroupOrExpression()

```
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupOrExpression.OptionalContentGroupOr↩
Expression (
            OptionalContentGroupExpression expression1,
            OptionalContentGroupExpression expression2,
            params OptionalContentGroupExpression[] expressions )
```

Create a new OptionalContentGroupOrExpression.

Parameters

expression1	The first argument.
expression2	The second argument.
expressions	Additional arguments.

Definition at line 352 of file OptionalContentGroups.cs.

7.122.3 Property Documentation

7.122.3.1 Operation

```
override OptionalContentGroupBooleanOperation VectSharp.PDF.OptionalContentGroups.Optional↩
ContentGroupOrExpression.Operation [get]
```

Definition at line 344 of file OptionalContentGroups.cs.

The documentation for this class was generated from the following file:

- VectSharp.PDF/OptionalContentGroups.cs

7.123 VectSharp.PDF.OptionalContentGroups.OptionalContentGroup↩ Settings Class Reference

Represents the settings for optional content groups (sometimes referred to as layers) in a PDF document.

Public Member Functions

- OptionalContentGroupSettings ()

  Create a new OptionalContentGroupSettings object.

Properties

- Dictionary< string, OptionalContentGroupExpression > Groups `[get, set]`

  A dictionary associating graphics tags to the corresponding visibility criteria.
- List< OptionalContentGroupTreeNode > OptionalContentGroupTree `[get, set]`

  A tree structure used by PDF viewers to present the optional content groups to the user. If this is set to `null` (the default), a flat tree (i.e. a list of all content groups) will be generated automatically. If this is empty but not `null`, no content groups are presented.
- List< List< OptionalContentGroup > > RadioButtonGroups `[get, set]`

  Radio button groups. For each group, at most one OptionalContentGroup can be enabled at the same time.

7.123.1 Detailed Description

Represents the settings for optional content groups (sometimes referred to as layers) in a PDF document.

Definition at line 540 of file OptionalContentGroups.cs.

7.123.2 Constructor & Destructor Documentation

7.123.2.1 OptionalContentGroupSettings()

```
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupSettings.OptionalContentGroupSettings
( )
```

Create a new OptionalContentGroupSettings object.

Definition at line 560 of file OptionalContentGroups.cs.

7.123.3 Property Documentation

7.123.3.1 Groups

```
Dictionary<string, OptionalContentGroupExpression> VectSharp.PDF.OptionalContentGroups.↩
OptionalContentGroupSettings.Groups  [get], [set]
```

A dictionary associating graphics tags to the corresponding visibility criteria.

Definition at line 545 of file OptionalContentGroups.cs.

7.123.3.2 OptionalContentGroupTree

```
List<OptionalContentGroupTreeNode> VectSharp.PDF.OptionalContentGroups.OptionalContentGroup↩
Settings.OptionalContentGroupTree  [get], [set]
```

A tree structure used by PDF viewers to present the optional content groups to the user. If this is set to `null` (the default), a flat tree (i.e. a list of all content groups) will be generated automatically. If this is empty but not `null`, no content groups are presented.

Definition at line 550 of file OptionalContentGroups.cs.

7.123.3.3 RadioButtonGroups

```
List<List<OptionalContentGroup> > VectSharp.PDF.OptionalContentGroups.OptionalContentGroup↩
Settings.RadioButtonGroups  [get], [set]
```

Radio button groups. For each group, at most one OptionalContentGroup can be enabled at the same time.

Definition at line 555 of file OptionalContentGroups.cs.

The documentation for this class was generated from the following file:

- VectSharp.PDF/OptionalContentGroups.cs

7.124 VectSharp.PDF.OptionalContentGroups.OptionalContentGroup↩ TreeLabel Class Reference

A label for an OptionalContentGroupTreeNode.

Public Types

- enum Type

  Types of label.

Public Member Functions

- OptionalContentGroupTreeLabel (string label)

  Create a new OptionalContentGroupTreeLabel representing a text string.
- OptionalContentGroupTreeLabel (OptionalContentGroup label)

  Create a new OptionalContentGroupTreeLabel representing an OptionalContentGroup.

Static Public Member Functions

- static implicit operator OptionalContentGroupTreeLabel (string label)

  Convert a `string` into an OptionalContentGroupTreeLabel.
- static implicit operator OptionalContentGroupTreeLabel (OptionalContentGroup label)

  Convert a OptionalContentGroup into an OptionalContentGroupTreeLabel.

Properties

- Type LabelType `[get]`

  The type of label.
- string LabelString `[get]`

  If this label's Type is Type.String, the text of the label.
- OptionalContentGroup LabelOptionalContentGroup `[get]`

  If this label's Type is Type.OptionalContentGroup, the OptionalContentGroup in the label.

7.124.1 Detailed Description

A label for an OptionalContentGroupTreeNode.

Definition at line 358 of file OptionalContentGroups.cs.

7.124.2 Member Enumeration Documentation

7.124.2.1 Type

```
enum VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeLabel.Type
```

Types of label.

Definition at line 363 of file OptionalContentGroups.cs.

7.124.3 Constructor & Destructor Documentation

7.124.3.1 OptionalContentGroupTreeLabel() [1/2]

```
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeLabel.OptionalContentGroupTree↩
Label (
            string label )
```

Create a new OptionalContentGroupTreeLabel representing a text string.

Parameters

label	The text for the label.

Definition at line 395 of file OptionalContentGroups.cs.

7.124.3.2 OptionalContentGroupTreeLabel() [2/2]

```
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeLabel.OptionalContentGroupTree↩
Label (
            OptionalContentGroup label )
```

Create a new OptionalContentGroupTreeLabel representing an OptionalContentGroup.

Parameters

label	The OptionalContentGroup in the label.

Definition at line 405 of file OptionalContentGroups.cs.

7.124.4 Member Function Documentation

7.124.4.1 operator OptionalContentGroupTreeLabel() [1/2]

```
static implicit VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeLabel.operator
OptionalContentGroupTreeLabel (
            OptionalContentGroup label ) [static]
```

Convert a OptionalContentGroup into an OptionalContentGroupTreeLabel.

Parameters

label	The OptionalContentGroup to convert.

7.124.4.2 operator OptionalContentGroupTreeLabel() [2/2]

```
static implicit VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeLabel.operator
OptionalContentGroupTreeLabel (
            string label ) [static]
```

Convert a `string` into an OptionalContentGroupTreeLabel.

Parameters

label	The `string` to convert.

7.124.5 Property Documentation

7.124.5.1 LabelOptionalContentGroup

```
OptionalContentGroup VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeLabel.Label↩
OptionalContentGroup [get]
```

If this label's Type is Type.OptionalContentGroup, the OptionalContentGroup in the label.

Definition at line 389 of file OptionalContentGroups.cs.

7.124.5.2 LabelString

```
string VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeLabel.LabelString  [get]
```

If this label's Type is Type.String, the text of the label.

Definition at line 384 of file OptionalContentGroups.cs.

7.124.5.3 LabelType

```
Type VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeLabel.LabelType  [get]
```

The type of label.

Definition at line 379 of file OptionalContentGroups.cs.

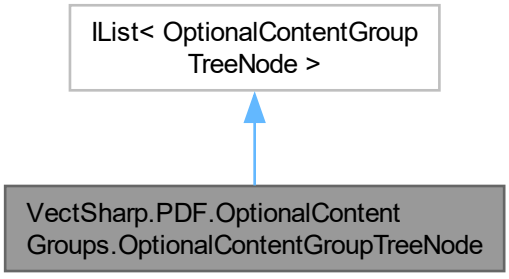
The documentation for this class was generated from the following file:

- VectSharp.PDF/OptionalContentGroups.cs

7.125 VectSharp.PDF.OptionalContentGroups.OptionalContentGroup↩ TreeNode Class Reference

Represents a node in an optional content group presentation tree.

Inheritance diagram for VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeNode:



Public Member Functions

- OptionalContentGroupTreeNode (OptionalContentGroupTreeLabel label)

  Create a new OptionalContentGroupTreeNode.
- int IndexOf (OptionalContentGroupTreeNode item)
- void Insert (int index, OptionalContentGroupTreeNode item)
- void RemoveAt (int index)
- void Add (OptionalContentGroupTreeNode item)
- void Clear ()
- bool Contains (OptionalContentGroupTreeNode item)
- void CopyTo (OptionalContentGroupTreeNode[ ] array, int arrayIndex)
- bool Remove (OptionalContentGroupTreeNode item)
- IEnumerator< OptionalContentGroupTreeNode > GetEnumerator ()

Static Public Member Functions

- static implicit operator OptionalContentGroupTreeNode (OptionalContentGroupTreeLabel label)

  Convert an OptionalContentGroupTreeLabel into a OptionalContentGroupTreeNode.
- static implicit operator OptionalContentGroupTreeNode (string label)

  Convert a `string` into a OptionalContentGroupTreeNode.
- static implicit operator OptionalContentGroupTreeNode (OptionalContentGroup label)

  Convert an OptionalContentGroup into a OptionalContentGroupTreeNode.

Properties

- OptionalContentGroupTreeLabel Label `[get, set]`

  Label for the node. This can be a text string or an OptionalContentGroup.
- List< OptionalContentGroupTreeNode > Children `[get, set]`

  Children for this node.
- int Count `[get]`
- bool IsReadOnly `[get]`
- OptionalContentGroupTreeNode this[int index] `[get, set]`

7.125.1 Detailed Description

Represents a node in an optional content group presentation tree.

Definition at line 428 of file OptionalContentGroups.cs.

7.125.2 Constructor & Destructor Documentation

7.125.2.1 OptionalContentGroupTreeNode()

```
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeNode.OptionalContentGroupTreeNode
(
            OptionalContentGroupTreeLabel label )
```

Create a new OptionalContentGroupTreeNode.

Parameters

label	The label for the node. This can be a text string or an OptionalContentGroup.

Definition at line 453 of file OptionalContentGroups.cs.

7.125.3 Member Function Documentation

7.125.3.1 Add()

```
void VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeNode.Add (
            OptionalContentGroupTreeNode item )
```

Definition at line 495 of file OptionalContentGroups.cs.

7.125.3.2 Clear()

```
void VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeNode.Clear ( )
```

Definition at line 501 of file OptionalContentGroups.cs.

7.125.3.3 Contains()

```
bool VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeNode.Contains (
            OptionalContentGroupTreeNode item )
```

Definition at line 507 of file OptionalContentGroups.cs.

7.125.3.4 CopyTo()

```
void VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeNode.CopyTo (
            OptionalContentGroupTreeNode[] array,
            int arrayIndex )
```

Definition at line 513 of file OptionalContentGroups.cs.

7.125.3.5 GetEnumerator()

```
IEnumerator< OptionalContentGroupTreeNode > VectSharp.PDF.OptionalContentGroups.Optional↩
ContentGroupTreeNode.GetEnumerator ( )
```

Definition at line 525 of file OptionalContentGroups.cs.

7.125.3.6 IndexOf()

```
int VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeNode.IndexOf (
            OptionalContentGroupTreeNode item )
```

Definition at line 477 of file OptionalContentGroups.cs.

7.125.3.7 Insert()

```
void VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeNode.Insert (
            int index,
            OptionalContentGroupTreeNode item )
```

Definition at line 483 of file OptionalContentGroups.cs.

7.125.3.8 operator OptionalContentGroupTreeNode() [1/3]

```
static implicit VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeNode.operator
OptionalContentGroupTreeNode (
            OptionalContentGroup label )  [static]
```

Convert an OptionalContentGroup into a OptionalContentGroupTreeNode.

Parameters

label	The node label.

7.125.3.9 operator OptionalContentGroupTreeNode() [2/3]

```
static implicit VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeNode.operator
OptionalContentGroupTreeNode (
            OptionalContentGroupTreeLabel label )  [static]
```

Convert an OptionalContentGroupTreeLabel into a OptionalContentGroupTreeNode.

Parameters

label	The node label. This can be a text string or an OptionalContentGroup.

7.125.3.10 operator OptionalContentGroupTreeNode() [3/3]

```
static implicit VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeNode.operator
OptionalContentGroupTreeNode (
            string label )  [static]
```

Convert a `string` into a OptionalContentGroupTreeNode.

Parameters

label	The node label.

7.125.3.11 Remove()

```
bool VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeNode.Remove (
            OptionalContentGroupTreeNode item )
```

Definition at line 519 of file OptionalContentGroups.cs.

7.125.3.12 RemoveAt()

```
void VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeNode.RemoveAt (
            int index )
```

Definition at line 489 of file OptionalContentGroups.cs.

7.125.4 Property Documentation

7.125.4.1 Children

```
List<OptionalContentGroupTreeNode> VectSharp.PDF.OptionalContentGroups.OptionalContentGroup↩
TreeNode.Children  [get], [set]
```

Children for this node.

Definition at line 438 of file OptionalContentGroups.cs.

7.125.4.2 Count

```
int VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeNode.Count  [get]
```

Definition at line 441 of file OptionalContentGroups.cs.

7.125.4.3 IsReadOnly

```
bool VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeNode.IsReadOnly  [get]
```

Definition at line 444 of file OptionalContentGroups.cs.

7.125.4.4 Label

```
OptionalContentGroupTreeLabel VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTree↩
Node.Label  [get], [set]
```

Label for the node. This can be a text string or an OptionalContentGroup.

Definition at line 433 of file OptionalContentGroups.cs.

7.125.4.5 this[int index]

```
OptionalContentGroupTreeNode VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTree↩
Node.this[int index]  [get], [set]
```

Definition at line 447 of file OptionalContentGroups.cs.

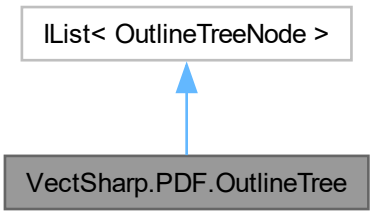
The documentation for this class was generated from the following file:

- VectSharp.PDF/OptionalContentGroups.cs

7.126 VectSharp.PDF.OutlineTree Class Reference

Represents an outline tree (table of contents) for a PDF document.

Inheritance diagram for VectSharp.PDF.OutlineTree:



Public Member Functions

- OutlineTree ()

  Create a new OutlineTree.
- int IndexOf (OutlineTreeNode item)
- void Insert (int index, OutlineTreeNode item)
- void RemoveAt (int index)
- void Add (OutlineTreeNode item)
- void Clear ()
- bool Contains (OutlineTreeNode item)
- void CopyTo (OutlineTreeNode[ ] array, int arrayIndex)
- bool Remove (OutlineTreeNode item)
- IEnumerator< OutlineTreeNode > GetEnumerator ()

Static Public Member Functions

- static OutlineTree CreateFromHeadings (List<(int level, string heading, string tag)> headings, int topLevel=1)

  Creates an OutlineTree from a list of headings, such as the one produced by VectSharp.Markdown.

Properties

- List< OutlineTreeNode > TopLevelItems `[get]`

  The top-level items in the outline.
- int Count `[get]`
- bool IsReadOnly `[get]`
- OutlineTreeNode this[int index] `[get, set]`

7.126.1 Detailed Description

Represents an outline tree (table of contents) for a PDF document.

Definition at line 26 of file OutlineTree.cs.

7.126.2 Constructor & Destructor Documentation

7.126.2.1 OutlineTree()

```
VectSharp.PDF.OutlineTree.OutlineTree ( )
```

Create a new OutlineTree.

Definition at line 45 of file OutlineTree.cs.

7.126.3 Member Function Documentation

7.126.3.1 Add()

```
void VectSharp.PDF.OutlineTree.Add (
            OutlineTreeNode item )
```

Definition at line 104 of file OutlineTree.cs.

7.126.3.2 Clear()

```
void VectSharp.PDF.OutlineTree.Clear ( )
```

Definition at line 110 of file OutlineTree.cs.

7.126.3.3 Contains()

```
bool VectSharp.PDF.OutlineTree.Contains (
            OutlineTreeNode item )
```

Definition at line 116 of file OutlineTree.cs.

7.126.3.4 CopyTo()

```
void VectSharp.PDF.OutlineTree.CopyTo (
            OutlineTreeNode[] array,
            int arrayIndex )
```

Definition at line 122 of file OutlineTree.cs.

7.126.3.5 CreateFromHeadings()

```
static OutlineTree VectSharp.PDF.OutlineTree.CreateFromHeadings (
            List<(int level, string heading, string tag)> headings,
            int topLevel = 1 )  [static]
```

Creates an OutlineTree from a list of headings, such as the one produced by VectSharp.Markdown.

Parameters

headings	The list of headings. Each element specifies a heading level (starting from 1), the text for the heading, and a tag for the corresponding object in the document.
topLevel	The heading level for top-level headings. Headings above this level will not be included in the outline tree. Setting this to a value higher than 1 is useful e.g. if the document has a title included as a level 1 heading.

Returns

An OutlineTree representation of the heading list.

Definition at line 56 of file OutlineTree.cs.

7.126.3.6 GetEnumerator()

```
IEnumerator< OutlineTreeNode > VectSharp.PDF.OutlineTree.GetEnumerator ( )
```

Definition at line 134 of file OutlineTree.cs.

7.126.3.7 IndexOf()

```
int VectSharp.PDF.OutlineTree.IndexOf (
            OutlineTreeNode item )
```

Definition at line 86 of file OutlineTree.cs.

7.126.3.8 Insert()

```
void VectSharp.PDF.OutlineTree.Insert (
            int index,
            OutlineTreeNode item )
```

Definition at line 92 of file OutlineTree.cs.

7.126.3.9 Remove()

```
bool VectSharp.PDF.OutlineTree.Remove (
            OutlineTreeNode item )
```

Definition at line 128 of file OutlineTree.cs.

7.126.3.10 RemoveAt()

```
void VectSharp.PDF.OutlineTree.RemoveAt (
            int index )
```

Definition at line 98 of file OutlineTree.cs.

7.126.4 Property Documentation

7.126.4.1 Count

```
int VectSharp.PDF.OutlineTree.Count  [get]
```

Definition at line 34 of file OutlineTree.cs.

7.126.4.2 IsReadOnly

```
bool VectSharp.PDF.OutlineTree.IsReadOnly  [get]
```

Definition at line 37 of file OutlineTree.cs.

7.126.4.3 this[int index]

```
OutlineTreeNode VectSharp.PDF.OutlineTree.this[int index]  [get], [set]
```

Definition at line 40 of file OutlineTree.cs.

7.126.4.4 TopLevelItems

```
List<OutlineTreeNode> VectSharp.PDF.OutlineTree.TopLevelItems  [get]
```

The top-level items in the outline.

Definition at line 31 of file OutlineTree.cs.

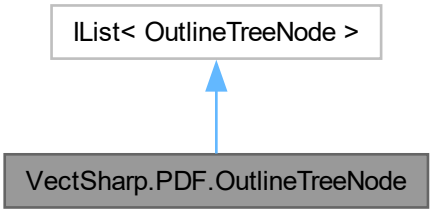
The documentation for this class was generated from the following file:

- VectSharp.PDF/OutlineTree.cs

7.127 VectSharp.PDF.OutlineTreeNode Class Reference

An item in an outline tree.

Inheritance diagram for VectSharp.PDF.OutlineTreeNode:



Public Member Functions

- OutlineTreeNode (string title, string destinationTag=null, Colour colour=default, bool bold=false, bool italic=false)

  Create a new OutlineTreeNode.
- int IndexOf (OutlineTreeNode item)
- void Insert (int index, OutlineTreeNode item)
- void RemoveAt (int index)
- void Add (OutlineTreeNode item)
- void Clear ()
- bool Contains (OutlineTreeNode item)
- void CopyTo (OutlineTreeNode[] array, int arrayIndex)
- bool Remove (OutlineTreeNode item)
- IEnumerator< OutlineTreeNode > GetEnumerator ()

Properties

- List< OutlineTreeNode > Children `[get]`

  Children of this item (can be empty).
- string Title `[get, set]`

  Text for this item.
- string DestinationTag `[get, set]`

  Tag of the graphics item that should be focused when the user clicks on the outline item.
- Colour Colour `[get, set]`

  Colour for the outline item (the alpha component is ignored).
- bool Bold `[get, set]`

  Indicates whether the outline item should be highlighted in bold or not.
- bool Italic `[get, set]`

  Indicates whether the outline item should be highlighted in italics or not.
- int Count `[get]`
- bool IsReadOnly `[get]`
- OutlineTreeNode this[int index] `[get, set]`

7.127.1 Detailed Description

An item in an outline tree.

Definition at line 148 of file OutlineTree.cs.

7.127.2 Constructor & Destructor Documentation

7.127.2.1 OutlineTreeNode()

```
VectSharp.PDF.OutlineTreeNode.OutlineTreeNode (
            string title,
            string destinationTag = null,
            Colour colour = default,
            bool bold = false,
            bool italic = false )
```

Create a new OutlineTreeNode.

Parameters

title	The text for the new outline item.
destinationTag	Tag of the graphics item that should be focused when the user clicks on the outline item.
colour	The colour for the outline item.
bold	Whether the outline item should be highlighted in bold or not.
italic	Whether the outline item should be highlighted in italics or not.

Definition at line 197 of file OutlineTree.cs.

7.127.3 Member Function Documentation

7.127.3.1 Add()

```
void VectSharp.PDF.OutlineTreeNode.Add (
            OutlineTreeNode item )
```

Definition at line 226 of file OutlineTree.cs.

7.127.3.2 Clear()

```
void VectSharp.PDF.OutlineTreeNode.Clear ( )
```

Definition at line 232 of file OutlineTree.cs.

7.127.3.3 Contains()

```
bool VectSharp.PDF.OutlineTreeNode.Contains (
            OutlineTreeNode item )
```

Definition at line 238 of file OutlineTree.cs.

7.127.3.4 CopyTo()

```
void VectSharp.PDF.OutlineTreeNode.CopyTo (
            OutlineTreeNode[] array,
            int arrayIndex )
```

Definition at line 244 of file OutlineTree.cs.

7.127.3.5 GetEnumerator()

```
IEnumerator< OutlineTreeNode > VectSharp.PDF.OutlineTreeNode.GetEnumerator ( )
```

Definition at line 256 of file OutlineTree.cs.

7.127.3.6 IndexOf()

```
int VectSharp.PDF.OutlineTreeNode.IndexOf (
            OutlineTreeNode item )
```

Definition at line 208 of file OutlineTree.cs.

7.127.3.7 Insert()

```
void VectSharp.PDF.OutlineTreeNode.Insert (
            int index,
            OutlineTreeNode item )
```

Definition at line 214 of file OutlineTree.cs.

7.127.3.8 Remove()

```
bool VectSharp.PDF.OutlineTreeNode.Remove (
            OutlineTreeNode item )
```

Definition at line 250 of file OutlineTree.cs.

7.127.3.9 RemoveAt()

```
void VectSharp.PDF.OutlineTreeNode.RemoveAt (
            int index )
```

Definition at line 220 of file OutlineTree.cs.

7.127.4 Property Documentation

7.127.4.1 Bold

```
bool VectSharp.PDF.OutlineTreeNode.Bold  [get], [set]
```

Indicates whether the outline item should be highlighted in bold or not.

Definition at line 173 of file OutlineTree.cs.

7.127.4.2 Children

```
List<OutlineTreeNode> VectSharp.PDF.OutlineTreeNode.Children  [get]
```

Children of this item (can be empty).

Definition at line 153 of file OutlineTree.cs.

7.127.4.3 Colour

```
Colour VectSharp.PDF.OutlineTreeNode.Colour  [get], [set]
```

Colour for the outline item (the alpha component is ignored).

Definition at line 168 of file OutlineTree.cs.

7.127.4.4 Count

```
int VectSharp.PDF.OutlineTreeNode.Count  [get]
```

Definition at line 181 of file OutlineTree.cs.

7.127.4.5 DestinationTag

```
string VectSharp.PDF.OutlineTreeNode.DestinationTag  [get], [set]
```

Tag of the graphics item that should be focused when the user clicks on the outline item.

Definition at line 163 of file OutlineTree.cs.

7.127.4.6 IsReadOnly

```
bool VectSharp.PDF.OutlineTreeNode.IsReadOnly  [get]
```

Definition at line 184 of file OutlineTree.cs.

7.127.4.7 Italic

```
bool VectSharp.PDF.OutlineTreeNode.Italic  [get], [set]
```

Indicates whether the outline item should be highlighted in italics or not.

Definition at line 178 of file OutlineTree.cs.

7.127.4.8 this[int index]

```
OutlineTreeNode VectSharp.PDF.OutlineTreeNode.this[int index]  [get], [set]
```

Definition at line 187 of file OutlineTree.cs.

7.127.4.9 Title

```
string VectSharp.PDF.OutlineTreeNode.Title  [get], [set]
```

Text for this item.

Definition at line 158 of file OutlineTree.cs.

The documentation for this class was generated from the following file:

- VectSharp.PDF/OutlineTree.cs

7.128 VectSharp.Page Class Reference

Represents a Graphics object with a width and height.

Public Member Functions

- Page (double width, double height)

  Create a new page.
- void Crop (Rectangle region, bool removeClippedGraphics=false, string tag=null)

  Translate and resize the Page so that it displays the specified region .
- void Crop (Point topLeft, Size size, bool removeClippedGraphics=false, string tag=null)

  Translate and resize the Page so that it displays the rectangle defined by topLeft and size .
- void Crop (string tag=null)

  Translate and resize the Page so that it displays the rectangle corresponding to the bounding box of its contents.

Properties

- double Width [get, set]

  Width of the page.
- double Height [get, set]

  Height of the page.
- Graphics Graphics [get, set]

  Graphics surface of the page.
- Colour Background = Colour.FromRgba(255, 255, 255, 0) [get, set]

  Background colour of the page.

7.128.1 Detailed Description

Represents a Graphics object with a width and height.

Definition at line 47 of file Document.cs.

7.128.2 Constructor & Destructor Documentation

7.128.2.1 Page()

```
VectSharp.Page.Page (
            double width,
            double height )
```

Create a new page.

Parameters

width	The width of the page.
height	The height of the page.

Definition at line 74 of file Document.cs.

7.128.3 Member Function Documentation

7.128.3.1 Crop() [1/3]

```
void VectSharp.Page.Crop (
            Point topLeft,
            Size size,
            bool removeClippedGraphics = false,
            string tag = null )
```

Translate and resize the Page so that it displays the rectangle defined by *topLeft* and *size* .

Parameters

topLeft	The top left corner of the area to include in the page.
size	The size of the area to include in the page.
removeClippedGraphics	If this is `true`, graphics actions that fall outside of the specified region are completely removed from the plot, otherwise they are just hidden.
tag	A tag to identify the transform.

Definition at line 101 of file Document.cs.

7.128.3.2 Crop() [2/3]

```
void VectSharp.Page.Crop (
            Rectangle region,
            bool removeClippedGraphics = false,
            string tag = null )
```

Translate and resize the Page so that it displays the specified *region* .

Parameters

region	The area to include in the page.
removeClippedGraphics	If this is `true`, graphics actions that fall outside of the specified region are completely removed from the plot, otherwise they are just hidden.
tag	A tag to identify the transform.

Definition at line 89 of file Document.cs.

7.128.3.3 Crop() [3/3]

```
void VectSharp.Page.Crop (
            string tag = null )
```

Translate and resize the Page so that it displays the rectangle corresponding to the bounding box of its contents.

Parameters

tag	A tag to identify the transform.

Definition at line 129 of file Document.cs.

7.128.4 Property Documentation

7.128.4.1 Background

```
Colour VectSharp.Page.Background = Colour.FromRgba(255, 255, 255, 0)  [get], [set]
```

Background colour of the page.

Definition at line 67 of file Document.cs.

7.128.4.2 Graphics

```
Graphics VectSharp.Page.Graphics  [get], [set]
```

Graphics surface of the page.

Definition at line 62 of file Document.cs.

7.128.4.3 Height

```
double VectSharp.Page.Height  [get], [set]
```

Height of the page.

Definition at line 57 of file Document.cs.

7.128.4.4 Width

```
double VectSharp.Page.Width  [get], [set]
```

Width of the page.

Definition at line 52 of file Document.cs.

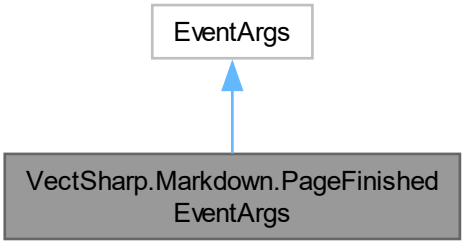
The documentation for this class was generated from the following file:

- VectSharp/Document.cs

7.129 VectSharp.Markdown.PageFinishedEventArgs Class Reference

EventArgs for the MarkdownRenderer.PageFinished event.

Inheritance diagram for VectSharp.Markdown.PageFinishedEventArgs:



Public Member Functions

- PageFinishedEventArgs (MarkdownContext markdownContext, Page page, Graphics graphics)

  Create a new PageFinishedEventArgs instance.

Properties

- MarkdownContext MarkdownContext `[get, set]`

  The current VectSharp.Markdown.MarkdownContext.
- Page Page `[get]`

  The finished VectSharp.Page.
- Graphics Graphics `[get, set]`

  The VectSharp.Graphics surface on which the page contents have been drawn.

7.129.1 Detailed Description

EventArgs for the MarkdownRenderer.PageFinished event.

Definition at line 271 of file MarkdownRenderer.Events.cs.

7.129.2 Constructor & Destructor Documentation

7.129.2.1 PageFinishedEventArgs()

```
VectSharp.Markdown.PageFinishedEventArgs.PageFinishedEventArgs (
            MarkdownContext markdownContext,
            Page page,
            Graphics graphics )
```

Create a new PageFinishedEventArgs instance.

Parameters

markdownContext	The current VectSharp.Markdown.MarkdownContext.
page	The finished VectSharp.Page.
graphics	The VectSharp.Graphics surface on which the page contents have been drawn.

Definition at line 294 of file MarkdownRenderer.Events.cs.

7.129.3 Property Documentation

7.129.3.1 Graphics

```
Graphics VectSharp.Markdown.PageFinishedEventArgs.Graphics  [get], [set]
```

The VectSharp.Graphics surface on which the page contents have been drawn.

Definition at line 286 of file MarkdownRenderer.Events.cs.

7.129.3.2 MarkdownContext

```
MarkdownContext VectSharp.Markdown.PageFinishedEventArgs.MarkdownContext  [get], [set]
```

The current VectSharp.Markdown.MarkdownContext.

Definition at line 276 of file MarkdownRenderer.Events.cs.

7.129.3.3 Page

```
Page VectSharp.Markdown.PageFinishedEventArgs.Page [get]
```

The finished VectSharp.Page.

Definition at line 281 of file MarkdownRenderer.Events.cs.

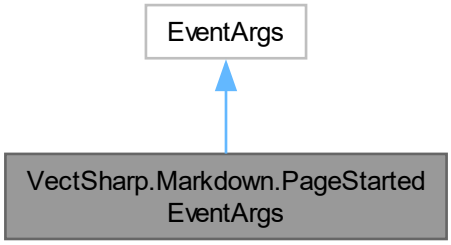
The documentation for this class was generated from the following file:

- VectSharp.Markdown/MarkdownRenderer.Events.cs

7.130 VectSharp.Markdown.PageStartedEventArgs Class Reference

EventArgs for the MarkdownRenderer.PageStarted event.

Inheritance diagram for VectSharp.Markdown.PageStartedEventArgs:



Public Member Functions

- PageStartedEventArgs (MarkdownContext markdownContext, Page page, Graphics graphics)

  Create a new PageStartedEventArgs instance.

Properties

- MarkdownContext MarkdownContext `[get, set]`

  The current VectSharp.Markdown.MarkdownContext.
- Page Page `[get]`

  The new empty VectSharp.Page.
- Graphics Graphics `[get, set]`

  The VectSharp.Graphics surface on which the page contents will be drawn.

7.130.1 Detailed Description

EventArgs for the MarkdownRenderer.PageStarted event.

Definition at line 237 of file MarkdownRenderer.Events.cs.

7.130.2 Constructor & Destructor Documentation

7.130.2.1 PageStartedEventArgs()

```
VectSharp.Markdown.PageStartedEventArgs.PageStartedEventArgs (
            MarkdownContext markdownContext,
            Page page,
            Graphics graphics )
```

Create a new PageStartedEventArgs instance.

Parameters

markdownContext	The current VectSharp.Markdown.MarkdownContext.
page	The new empty VectSharp.Page.
graphics	The VectSharp.Graphics surface on which the page contents will be drawn.

Definition at line 260 of file MarkdownRenderer.Events.cs.

7.130.3 Property Documentation

7.130.3.1 Graphics

```
Graphics VectSharp.Markdown.PageStartedEventArgs.Graphics  [get], [set]
```

The VectSharp.Graphics surface on which the page contents will be drawn.

Definition at line 252 of file MarkdownRenderer.Events.cs.

7.130.3.2 MarkdownContext

```
MarkdownContext VectSharp.Markdown.PageStartedEventArgs.MarkdownContext  [get], [set]
```

The current VectSharp.Markdown.MarkdownContext.

Definition at line 242 of file MarkdownRenderer.Events.cs.

7.130.3.3 Page

```
Page VectSharp.Markdown.PageStartedEventArgs.Page [get]
```

The new empty VectSharp.Page.

Definition at line 247 of file MarkdownRenderer.Events.cs.

The documentation for this class was generated from the following file:

- VectSharp.Markdown/MarkdownRenderer.Events.cs

7.131 VectSharp.TrueTypeFile.PairKerning Class Reference

Contains information describing how the position of two glyphs in a kerning pair should be altered.

Properties

- Point Glyph1Placement [get]
  This vector contains the displacement that should be applied to the first glyph of the pair.
- Point Glyph1Advance [get]
  This vector describes how the advance width of the first glyph should be altered.
- Point Glyph2Placement [get]
  This vector contains the displacement that should be applied to the second glyph of the pair.
- Point Glyph2Advance [get]
  This vector describes how the advance width of the second glyph should be altered.

7.131.1 Detailed Description

Contains information describing how the position of two glyphs in a kerning pair should be altered.

Definition at line 4442 of file TrueType.cs.

7.131.2 Property Documentation

7.131.2.1 Glyph1Advance

```
Point VectSharp.TrueTypeFile.PairKerning.Glyph1Advance [get]
```

This vector describes how the advance width of the first glyph should be altered.

Definition at line 4452 of file TrueType.cs.

7.131.2.2 Glyph1Placement

```
Point VectSharp.TrueTypeFile.PairKerning.Glyph1Placement [get]
```

This vector contains the displacement that should be applied to the first glyph of the pair.

Definition at line 4447 of file TrueType.cs.

7.131.2.3 Glyph2Advance

```
Point VectSharp.TrueTypeFile.PairKerning.Glyph2Advance [get]
```

This vector describes how the advance width of the second glyph should be altered.

Definition at line 4462 of file TrueType.cs.

7.131.2.4 Glyph2Placement

```
Point VectSharp.TrueTypeFile.PairKerning.Glyph2Placement [get]
```

This vector contains the displacement that should be applied to the second glyph of the pair.

Definition at line 4457 of file TrueType.cs.

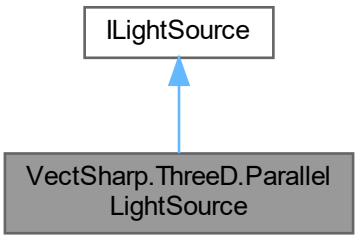
The documentation for this class was generated from the following file:

- VectSharp/TrueType.cs

7.132 VectSharp.ThreeD.ParallelLightSource Class Reference

Represents a parallel light source.

Inheritance diagram for VectSharp.ThreeD.ParallelLightSource:



Public Member Functions

- ParallelLightSource (double intensity, NormalizedVector3D direction)

  Creates a new ParallelLightSource instance.
- LightIntensity GetLightAt (Point3D point)

  Computes the light intensity at the specified point, without taking into account any obstructions.

Parameters

point	*The Point3DElement at which the light intensity should be computed.*

Returns

- double GetObstruction (Point3D point, IEnumerable< Triangle3DElement > shadowingTriangles)

  Determines the amount of obstruction of the light that results at point due to the specified shadowingTriangles .

Parameters

point	*The Point3D at which the obstruction should be computed.*
shadowingTriangles	*A collection of Triangle3DElement casting shadows.*

Returns

1 if the light is completely obstructed, 0 if the light is completely visible, a value between these if the light is partially obstructed.

Properties

- double Intensity `[get, set]`

  The intensity of the light.
- NormalizedVector3D Direction `[get]`

  The direction along which the light travels.
- NormalizedVector3D ReverseDirection `[get]`

  The reverse of Direction.
- bool CastsShadow = true `[get, set]`

  Determines whether the light casts a shadow or not.

7.132.1 Detailed Description

Represents a parallel light source.

Definition at line 126 of file Lights.cs.

7.132.2 Constructor & Destructor Documentation

7.132.2.1 ParallelLightSource()

```
VectSharp.ThreeD.ParallelLightSource.ParallelLightSource (
            double intensity,
            NormalizedVector3D direction )
```

Creates a new ParallelLightSource instance.

Parameters

intensity	The intensity of the light.
direction	The direction along which the light travels.

Definition at line 151 of file Lights.cs.

7.132.3 Member Function Documentation

7.132.3.1 GetLightAt()

```
LightIntensity VectSharp.ThreeD.ParallelLightSource.GetLightAt (
            Point3D point )
```

Computes the light intensity at the specified point, without taking into account any obstructions.

Parameters

point	The Point3DElement at which the light intensity should be computed.

Returns

Implements VectSharp.ThreeD.ILightSource.

Definition at line 159 of file Lights.cs.

7.132.3.2 GetObstruction()

```
double VectSharp.ThreeD.ParallelLightSource.GetObstruction (
            Point3D point,
            IEnumerable< Triangle3DElement > shadowingTriangles )
```

Determines the amount of obstruction of the light that results at *point* due to the specified *shadowingTriangles* .

Parameters

point	The Point3D at which the obstruction should be computed.
shadowingTriangles	A collection of Triangle3DElement casting shadows.

Returns

1 if the light is completely obstructed, 0 if the light is completely visible, a value between these if the light is partially obstructed.

Implements VectSharp.ThreeD.ILightSource.

Definition at line 165 of file Lights.cs.

7.132.4 Property Documentation

7.132.4.1 CastsShadow

```
bool VectSharp.ThreeD.ParallelLightSource.CastsShadow = true  [get], [set]
```

Determines whether the light casts a shadow or not.

Implements VectSharp.ThreeD.ILightSource.

Definition at line 144 of file Lights.cs.

7.132.4.2 Direction

```
NormalizedVector3D VectSharp.ThreeD.ParallelLightSource.Direction  [get]
```

The direction along which the light travels.

Definition at line 136 of file Lights.cs.

7.132.4.3 Intensity

```
double VectSharp.ThreeD.ParallelLightSource.Intensity  [get], [set]
```

The intensity of the light.

Definition at line 131 of file Lights.cs.

7.132.4.4 ReverseDirection

```
NormalizedVector3D VectSharp.ThreeD.ParallelLightSource.ReverseDirection  [get]
```

The reverse of Direction.

Definition at line 141 of file Lights.cs.

The documentation for this class was generated from the following file:

- VectSharp.ThreeD/Lights.cs

7.133 VectSharp.SVG.Parser Class Reference

Contains methods to read an SVG image file.

Static Public Member Functions

- static Page ParseSVGURI (string uri, bool ignored=false)

  Parses an SVG image URI.
- static Page FromString (string svgSource)

  Parses SVG source into a Page containing the image represented by the code.
- static Page FromFile (string fileName)

  Parses an SVG image file into a Page containing the image.
- static Page FromStream (Stream svgSourceStream)

  Parses an stream containing SVG source code into a Page containing the image represented by the code.

Static Public Attributes

- static Func< string, bool, Page > ParseImageURI

  A function that takes as input an image URI and a boolean value indicating whether the image should be interpolated, and returns a Page object containing the image. By default, this is equal to ParseSVGURI, i.e. it is only able to parse SVG images. If you wish to enable the parsing of other formats, you should install the "VectSharp.MuPDFUtils" NuGet package and enable the parser in your program by doing something like:

7.133.1 Detailed Description

Contains methods to read an SVG image file.

Definition at line 33 of file SVGParser.cs.

7.133.2 Member Function Documentation

7.133.2.1 FromFile()

```
static Page VectSharp.SVG.Parser.FromFile (
            string fileName )  [static]
```

Parses an SVG image file into a Page containing the image.

Parameters

fileName	The path to the SVG image file.

Returns

A Page containing the image represented by the file.

Definition at line 200 of file SVGParser.cs.

7.133.2.2 FromStream()

```
static Page VectSharp.SVG.Parser.FromStream (
            Stream svgSourceStream )  [static]
```

Parses an stream containing SVG source code into a Page containing the image represented by the code.

Parameters

svgSourceStream	The stream containing SVG source code.

Returns

A Page containing the image represented by the *svgSourceStream* .

Definition at line 210 of file SVGParser.cs.

7.133.2.3 FromString()

```
static Page VectSharp.SVG.Parser.FromString (
            string svgSource )  [static]
```

Parses SVG source into a Page containing the image represented by the code.

Parameters

svgSource	The SVG source code.

Returns

A Page containing the image represented by the *svgSource* .

Definition at line 116 of file SVGParser.cs.

7.133.2.4 ParseSVGURI()

```
static Page VectSharp.SVG.Parser.ParseSVGURI (
            string uri,
            bool ignored = false )  [static]
```

Parses an SVG image URI.

Parameters

uri	The image URI to parse.
ignored	This value is ignored and is only needed for compatibility.

Returns

A Page containing the parsed SVG image, or null.

Definition at line 54 of file SVGParser.cs.

7.133.3 Member Data Documentation

7.133.3.1 ParseImageURI

```
Func<string, bool, Page> VectSharp.SVG.Parser.ParseImageURI  [static]
```

A function that takes as input an image URI and a boolean value indicating whether the image should be interpolated, and returns a Page object containing the image. By default, this is equal to ParseSVGURI, i.e. it is only able to parse SVG images. If you wish to enable the parsing of other formats, you should install the "VectSharp.MuPDFUtils" NuGet package and enable the parser in your program by doing something like:

```
VectSharp.SVG.Parser.ParseImageURI = VectSharp.MuPDFUtils.ImageURIParser.Parser(VectShar
```

Definition at line 46 of file SVGParser.cs.

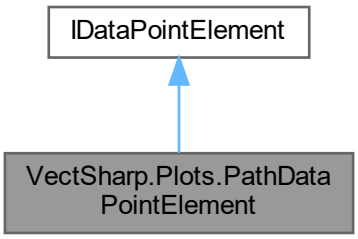
The documentation for this class was generated from the following file:

- VectSharp.SVG/SVGParser.cs

7.134 VectSharp.Plots.PathDataPointElement Class Reference

A symbol defined by a GraphicsPath.

Inheritance diagram for VectSharp.Plots.PathDataPointElement:



Public Member Functions

- void Plot (Graphics target, PlotElementPresentationAttributes presentationAttributes, string tag)

  Draw the symbol on the plot.

Parameters

target	*The Graphics object on which to draw. It is assumed that it has been transformed so that the symbol can be drawn centred at (0, 0)*
presentationAttributes	*Presentation attributes determining the appearance of the symbol.*
tag	*A tag to identify the symbol in the plot.*

- PathDataPointElement ()

  Create a new PathDataPointElement instance representing a circle.
- PathDataPointElement (GraphicsPath path)

  Create a new PathDataPointElement instance with the specified path .

Properties

- GraphicsPath Path = new GraphicsPath().Arc(0, 0, 1, 0, 2 * Math.PI).Close() `[get, set]`

  The GraphicsPath that constitutes the symbol (by default, a circle).

7.134.1 Detailed Description

A symbol defined by a GraphicsPath.

Definition at line 40 of file DataPoints.cs.

7.134.2 Constructor & Destructor Documentation

7.134.2.1 PathDataPointElement() [1/2]

```
VectSharp.Plots.PathDataPointElement.PathDataPointElement ( )
```

Create a new PathDataPointElement instance representing a circle.

Definition at line 79 of file DataPoints.cs.

7.134.2.2 PathDataPointElement() [2/2]

```
VectSharp.Plots.PathDataPointElement.PathDataPointElement (
            GraphicsPath path )
```

Create a new PathDataPointElement instance with the specified *path* .

Definition at line 87 of file DataPoints.cs.

7.134.3 Member Function Documentation

7.134.3.1 Plot()

```
void VectSharp.Plots.PathDataPointElement.Plot (
            Graphics target,
            PlotElementPresentationAttributes presentationAttributes,
            string tag )
```

Draw the symbol on the plot.

Parameters

target	The Graphics object on which to draw. It is assumed that it has been transformed so that the symbol can be drawn centred at (0, 0)
presentationAttributes	Presentation attributes determining the appearance of the symbol.
tag	A tag to identify the symbol in the plot.

Implements VectSharp.Plots.IDataPointElement.

Definition at line 48 of file DataPoints.cs.

7.134.4 Property Documentation

7.134.4.1 Path

```
GraphicsPath VectSharp.Plots.PathDataPointElement.Path = new GraphicsPath().Arc(0, 0, 1, 0, 2
* Math.PI).Close()  [get], [set]
```

The GraphicsPath that constitutes the symbol (by default, a circle).

Definition at line 45 of file DataPoints.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/DataPoints.cs

7.135 VectSharp.PDF.PDFContextInterpreter Class Reference

Contains methods to render a Document as a PDF document.

Classes

- class FilterOption

  Determines how and whether image filters are rasterised.

Public Types

- enum TextOptions

  Defines whether the used fonts should be included in the file.

Static Public Member Functions

- static PDFDocument CreatePDFDocument (this Document document, TextOptions textOption=TextOptions.SubsetFonts, bool compressStreams=true, Dictionary< string, string > linkDestinations=null, OutlineTree outline=null, PDFMetadata metadata=default, FilterOption filterOption=default, OptionalContentGroupSettings optionalContentGroupSettings=default, AnnotationStyleCollection annotationStyles=default)

  Convert the document to a PDFDocument representation.
- static void SaveAsPDF (this Document document, string fileName, TextOptions textOption=TextOptions.SubsetFonts, bool compressStreams=true, Dictionary< string, string > linkDestinations=null, OutlineTree outline=null, PDFMetadata metadata=default, FilterOption filterOption=default, OptionalContentGroupSettings optionalContentGroupSettings=default, AnnotationStyleCollection annotationStyles=default)

  Save the document to a PDF file.
- static void SaveAsPDF (this Document document, Stream stream, TextOptions textOption=TextOptions.SubsetFonts, bool compressStreams=true, Dictionary< string, string > linkDestinations=null, OutlineTree outline=null, PDFMetadata metadata=default, FilterOption filterOption=default, OptionalContentGroupSettings optionalContentGroupSettings=default, AnnotationStyleCollection annotationStyles=default)

  Save the document to a PDF stream.

7.135.1 Detailed Description

Contains methods to render a Document as a PDF document.

Definition at line 86 of file PDFContextInterpreter.cs.

7.135.2 Member Enumeration Documentation

7.135.2.1 TextOptions

```
enum VectSharp.PDF.PDFContextInterpreter.TextOptions
```

Defines whether the used fonts should be included in the file.

Definition at line 91 of file PDFContextInterpreter.cs.

7.135.3 Member Function Documentation

7.135.3.1 CreatePDFDocument()

```
static PDFDocument VectSharp.PDF.PDFContextInterpreter.CreatePDFDocument (
            this Document document,
            TextOptions textOption = TextOptions.SubsetFonts,
            bool compressStreams = true,
            Dictionary< string, string > linkDestinations = null,
            OutlineTree outline = null,
            PDFMetadata metadata = default,
            FilterOption filterOption = default,
            OptionalContentGroupSettings optionalContentGroupSettings = default,
            AnnotationStyleCollection annotationStyles = default )  [static]
```

Convert the document to a PDFDocument representation.

Parameters

document	The Document to convert.
textOption	Defines whether the used fonts should be included in the PDF document.
compressStreams	Indicates whether the streams in the PDF document should be compressed.
linkDestinations	A dictionary associating element tags to link targets. If this is provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to the destination specified in the dictionary. If the destination starts with a hash (#), it is interpreted as the tag of another object in the current document; otherwise, it is interpreted as an external URI.
outline	Document outline (table of contents).
metadata	Document metadata. Use PDFMetadata.NoMetadata() if you do not wish to include metadata in the document.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.
optionalContentGroupSettings	Settings for optional content groups (layers).
annotationStyles	Annotation appearance styles.

Definition at line 748 of file PDFContextInterpreter.cs.

7.135.3.2 SaveAsPDF() [1/2]

```
static void VectSharp.PDF.PDFContextInterpreter.SaveAsPDF (
            this Document document,
            Stream stream,
            TextOptions textOption = TextOptions.SubsetFonts,
            bool compressStreams = true,
            Dictionary< string, string > linkDestinations = null,
            OutlineTree outline = null,
            PDFMetadata metadata = default,
            FilterOption filterOption = default,
            OptionalContentGroupSettings optionalContentGroupSettings = default,
            AnnotationStyleCollection annotationStyles = default )  [static]
```

Save the document to a PDF stream.

Parameters

document	The Document to save.
stream	The stream to which the PDF data will be written.
textOption	Defines whether the used fonts should be included in the file.
compressStreams	Indicates whether the streams in the PDF file should be compressed.
linkDestinations	A dictionary associating element tags to link targets. If this is provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to the destination specified in the dictionary. If the destination starts with a hash (#), it is interpreted as the tag of another object in the current document; otherwise, it is interpreted as an external URI.
outline	Document outline (table of contents).
metadata	Document metadata. Use PDFMetadata.NoMetadata() if you do not wish to include metadata in the document.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.
optionalContentGroupSettings	Settings for optional content groups (layers).
annotationStyles	Annotation appearance styles.

Definition at line 899 of file PDFContextInterpreter.cs.

7.135.3.3 SaveAsPDF() [2/2]

```
static void VectSharp.PDF.PDFContextInterpreter.SaveAsPDF (
            this Document document,
            string fileName,
            TextOptions textOption = TextOptions.SubsetFonts,
            bool compressStreams = true,
            Dictionary< string, string > linkDestinations = null,
            OutlineTree outline = null,
```

```
PDFMetadata metadata = default,
FilterOption filterOption = default,
OptionalContentGroupSettings optionalContentGroupSettings = default,
AnnotationStyleCollection annotationStyles = default )  [static]
```

Save the document to a PDF file.

Parameters

document	The Document to save.
fileName	The full path to the file to save. If it exists, it will be overwritten.
textOption	Defines whether the used fonts should be included in the file.
compressStreams	Indicates whether the streams in the PDF file should be compressed.
linkDestinations	A dictionary associating element tags to link targets. If this is provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to the destination specified in the dictionary. If the destination starts with a hash (#), it is interpreted as the tag of another object in the current document; otherwise, it is interpreted as an external URI.
outline	Document outline (table of contents).
metadata	Document metadata. Use PDFMetadata.NoMetadata() if you do not wish to include metadata in the document.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.
optionalContentGroupSettings	Settings for optional content groups (layers).
annotationStyles	Annotation appearance styles.

Definition at line 878 of file PDFContextInterpreter.cs.

The documentation for this class was generated from the following file:

- VectSharp.PDF/PDFContextInterpreter.cs

7.136 VectSharp.PDF.PDFDocument Class Reference

A low-level representation of a PDF document.

Public Member Functions

- void Write (Stream stream, Encoding encoding=default)

  Write the PDF document to the specified Stream.
- void Write (string file, Encoding encoding=default)

  Write the PDF document to the specified file.
- PDFDocument (IList< PDFReferenceableObject > pdfObjects)

  Create a new PDFDocument containing the specified pdfObjects .

Static Public Attributes

- static readonly Encoding DefaultEncoding = Encoding.GetEncoding("ISO-8859-1")

  Default document encoding.

Properties

- IList< PDFReferenceableObject > Contents [get]

  Objects contained in the PDF document.
- string PDFVersion = "1.4" [get, set]

  PDF specification version. This is only included in the header and not enforced.
- PDFCatalog Catalog [get, set]

  Gets or sets the PDF catalog object.
- PDFDocumentInfo Info [get, set]

  Gets or sets the PDF document info object.
- List< PDFPage > Pages [get]

  Gets the document page list.

7.136.1 Detailed Description

A low-level representation of a PDF document.

Definition at line 29 of file PDFDocument.cs.

7.136.2 Constructor & Destructor Documentation

7.136.2.1 PDFDocument()

```
VectSharp.PDF.PDFDocument.PDFDocument (
            IList< PDFReferenceableObject > pdfObjects )
```

Create a new PDFDocument containing the specified *pdfObjects* .

Parameters

pdfObjects	The objects contained in the PDF document.

Definition at line 270 of file PDFDocument.cs.

7.136.3 Member Function Documentation

7.136.3.1 Write() [1/2]

```
void VectSharp.PDF.PDFDocument.Write (
            Stream stream,
            Encoding encoding = default )
```

Write the PDF document to the specified Stream.

Parameters

stream	The stream on which the PDF document should be written.
encoding	The document encoding (default: ISO 8859-1).

Definition at line 201 of file PDFDocument.cs.

7.136.3.2 Write() [2/2]

```
void VectSharp.PDF.PDFDocument.Write (
            string file,
            Encoding encoding = default )
```

Write the PDF document to the specified file.

Parameters

file	The file on which the PDF document should be written.
encoding	The document encoding (default: ISO 8859-1).

Definition at line 258 of file PDFDocument.cs.

7.136.4 Member Data Documentation

7.136.4.1 DefaultEncoding

```
readonly Encoding VectSharp.PDF.PDFDocument.DefaultEncoding = Encoding.GetEncoding("ISO-8859-1")
[static]
```

Default document encoding.

Definition at line 194 of file PDFDocument.cs.

7.136.5 Property Documentation

7.136.5.1 Catalog

```
PDFCatalog VectSharp.PDF.PDFDocument.Catalog  [get], [set]
```

Gets or sets the PDF catalog object.

Definition at line 44 of file PDFDocument.cs.

7.136.5.2 Contents

```
IList<PDFReferenceableObject> VectSharp.PDF.PDFDocument.Contents  [get]
```

Objects contained in the PDF document.

Definition at line 34 of file PDFDocument.cs.

7.136.5.3 Info

```
PDFDocumentInfo VectSharp.PDF.PDFDocument.Info  [get], [set]
```

Gets or sets the PDF document info object.

Definition at line 112 of file PDFDocument.cs.

7.136.5.4 Pages

```
List<PDFPage> VectSharp.PDF.PDFDocument.Pages  [get]
```

Gets the document page list.

Definition at line 186 of file PDFDocument.cs.

7.136.5.5 PDFVersion

```
string VectSharp.PDF.PDFDocument.PDFVersion = "1.4"  [get], [set]
```

PDF specification version. This is only included in the header and not enforced.

Definition at line 39 of file PDFDocument.cs.

The documentation for this class was generated from the following file:

- VectSharp.PDF/PDFDocument.cs

7.137 VectSharp.PDF.PDFMetadata Class Reference

Represents metadata for a PDF document.

Public Member Functions

- PDFMetadata ()

  Create a new PDF metadata object.

Static Public Member Functions

- static PDFMetadata NoMetadata ()

  Returns an object that does not include any metadata in the PDF file.

Properties

- bool ExcludeMetadata = false `[get, set]`

  Indicates that no metadata should be included in PDF document.
- string Title `[get, set]`

  The document title.
- string Author = Environment.UserName `[get, set]`

  The person who created the document.
- string Subject `[get, set]`

  The subject of the document.
- string Keywords `[get, set]`

  Keywords associated with the document.
- string Creator = "VectSharp v" + typeof(Document).Assembly.GetName().Version.ToString(3) `[get, set]`

  The program used to create the original version of the document (e.g., VectSharp).
- string Producer = "VectSharp.PDF v" + typeof(PDFMetadata).Assembly.GetName().Version.ToString(3) `[get, set]`

  The program used to create the PDF document (e.g., VectSharp.PDF).
- DateTime? CreationDate = DateTime.Now `[get, set]`

  The date when the PDF file was created.
- TimeZoneInfo CreationDateTimeZone = TimeZoneInfo.Local `[get, set]`

  The time zone where the PDF file was created.
- DateTime? ModificationDate = DateTime.Now `[get, set]`

  The date when the PDF file was last modified.
- TimeZoneInfo ModificationDateTimeZone = TimeZoneInfo.Local `[get, set]`

  The time zone where the PDF file was last modified.
- Dictionary< string, object > CustomProperties `[get, set]`

  Custom metadata properties to include in the PDF document.

7.137.1 Detailed Description

Represents metadata for a PDF document.

Definition at line 27 of file PDFMetadata.cs.

7.137.2 Constructor & Destructor Documentation

7.137.2.1 PDFMetadata()

```
VectSharp.PDF.PDFMetadata.PDFMetadata ( )
```

Create a new PDF metadata object.

Definition at line 91 of file PDFMetadata.cs.

7.137.3 Member Function Documentation

7.137.3.1 NoMetadata()

```
static PDFMetadata VectSharp.PDF.PDFMetadata.NoMetadata ( )  [static]
```

Returns an object that does not include any metadata in the PDF file.

Returns

An object that does not include any metadata in the PDF file.

Definition at line 180 of file PDFMetadata.cs.

7.137.4 Property Documentation

7.137.4.1 Author

```
string VectSharp.PDF.PDFMetadata.Author = Environment.UserName  [get], [set]
```

The person who created the document.

Definition at line 41 of file PDFMetadata.cs.

7.137.4.2 CreationDate

```
DateTime?  VectSharp.PDF.PDFMetadata.CreationDate = DateTime.Now  [get], [set]
```

The date when the PDF file was created.

Definition at line 66 of file PDFMetadata.cs.

7.137.4.3 CreationDateTimeZone

```
TimeZoneInfo VectSharp.PDF.PDFMetadata.CreationDateTimeZone = TimeZoneInfo.Local  [get], [set]
```

The time zone where the PDF file was created.

Definition at line 71 of file PDFMetadata.cs.

7.137.4.4 Creator

```
string VectSharp.PDF.PDFMetadata.Creator = "VectSharp v" + typeof(Document).Assembly.Get↩
Name().Version.ToString(3)  [get], [set]
```

The program used to create the original version of the document (e.g., VectSharp).

Definition at line 56 of file PDFMetadata.cs.

7.137.4.5 CustomProperties

```
Dictionary<string, object> VectSharp.PDF.PDFMetadata.CustomProperties  [get], [set]
```

Custom metadata properties to include in the PDF document.

Definition at line 86 of file PDFMetadata.cs.

7.137.4.6 ExcludeMetadata

```
bool VectSharp.PDF.PDFMetadata.ExcludeMetadata = false  [get], [set]
```

Indicates that no metadata should be included in PDF document.

Definition at line 32 of file PDFMetadata.cs.

7.137.4.7 Keywords

```
string VectSharp.PDF.PDFMetadata.Keywords  [get], [set]
```

Keywords associated with the document.

Definition at line 51 of file PDFMetadata.cs.

7.137.4.8 ModificationDate

```
DateTime?  VectSharp.PDF.PDFMetadata.ModificationDate = DateTime.Now  [get], [set]
```

The date when the PDF file was last modified.

Definition at line 76 of file PDFMetadata.cs.

7.137.4.9 ModificationDateTimeZone

```
TimeZoneInfo VectSharp.PDF.PDFMetadata.ModificationDateTimeZone = TimeZoneInfo.Local  [get],
[set]
```

The time zone where the PDF file was last modified.

Definition at line 81 of file PDFMetadata.cs.

7.137.4.10 Producer

```
string VectSharp.PDF.PDFMetadata.Producer = "VectSharp.PDF v" + typeof(PDFMetadata).Assembly.↩
GetName().Version.ToString(3)  [get], [set]
```

The program used to create the PDF document (e.g., VectSharp.PDF).

Definition at line 61 of file PDFMetadata.cs.

7.137.4.11 Subject

```
string VectSharp.PDF.PDFMetadata.Subject  [get], [set]
```

The subject of the document.

Definition at line 46 of file PDFMetadata.cs.

7.137.4.12 Title

```
string VectSharp.PDF.PDFMetadata.Title  [get], [set]
```

The document title.

Definition at line 36 of file PDFMetadata.cs.

The documentation for this class was generated from the following file:

- VectSharp.PDF/PDFMetadata.cs

7.138 VectSharp.MuPDFUtils.PDFParser Class Reference

Contains methods to import PDF documents as vectors.

Static Public Member Functions

- static Page FromFile (string fileName, int pageNumber, SVGCreationOptions.TextOption textOption=SVGCreation↩ Options.TextOption.TextAsPath, bool includeAnnotations=true)

  Import the specified page from a document.
- static Page FromFile (string fileName, int pageNumber, out Dictionary< string, string > linkDestinations, SVGCreationOptions.TextOption textOption=SVGCreationOptions.TextOption.TextAsPath, bool include↩ Annotations=true)

  Import the specified page from a document.
- static Page FromStream (Stream stream, int pageNumber, InputFileTypes fileType=InputFileTypes.PDF, SVGCreationOptions.TextOption textOption=SVGCreationOptions.TextOption.TextAsPath, bool include↩ Annotations=true)

  Import the specified page from a document.
- static Page FromStream (Stream stream, int pageNumber, out Dictionary< string, string > linkDestinations, InputFileTypes fileType=InputFileTypes.PDF, SVGCreationOptions.TextOption textOption=SVGCreation↩ Options.TextOption.TextAsPath, bool includeAnnotations=true)

  Import the specified page from a document.
- static Document FromFile (string fileName, SVGCreationOptions.TextOption textOption=SVGCreation↩ Options.TextOption.TextAsPath, bool includeAnnotations=true)

  Import all pages from a document.
- static Document FromFile (string fileName, out Dictionary< string, string > linkDestinations, SVGCreation↩ Options.TextOption textOption=SVGCreationOptions.TextOption.TextAsPath, bool includeAnnotations=true)

  Import all pages from a document.
- static Document FromStream (Stream stream, InputFileTypes fileType=InputFileTypes.PDF, SVGCreation↩ Options.TextOption textOption=SVGCreationOptions.TextOption.TextAsPath, bool includeAnnotations=true)

  Import all pages from a document.
- static Document FromStream (Stream stream, out Dictionary< string, string > linkDestinations, InputFile↩ Types fileType=InputFileTypes.PDF, SVGCreationOptions.TextOption textOption=SVGCreationOptions.Text↩ Option.TextAsPath, bool includeAnnotations=true)

  Import all pages from a document.

7.138.1 Detailed Description

Contains methods to import PDF documents as vectors.

Definition at line 29 of file PDFParser.cs.

7.138.2 Member Function Documentation

7.138.2.1 FromFile() [1/4]

```
static Page VectSharp.MuPDFUtils.PDFParser.FromFile (
            string fileName,
            int pageNumber,
            out Dictionary< string, string > linkDestinations,
            SVGCreationOptions.TextOption textOption = SVGCreationOptions.TextOption.TextAsPath,
            bool includeAnnotations = true )  [static]
```

Import the specified page from a document.

Parameters

fileName	The path to the document file.
pageNumber	The page number (starting at 0).
linkDestinations	When this method returns, this variable will contain a dictionary used to associate graphic action tags to hyperlinks. This can be used to enable such links when rendering the Page to a file.
textOption	This parameter determines whether text elements are rendered as paths (which provides more accurate results) or as text.
includeAnnotations	Whether annotations (e.g., signatures) should be included in the imported document.

Returns

A Page containing a representation of the specified page from the document.

Definition at line 185 of file PDFParser.cs.

7.138.2.2 FromFile() [2/4]

```
static Page VectSharp.MuPDFUtils.PDFParser.FromFile (
            string fileName,
            int pageNumber,
            SVGCreationOptions.TextOption textOption = SVGCreationOptions.TextOption.TextAsPath,
            bool includeAnnotations = true )  [static]
```

Import the specified page from a document.

Parameters

fileName	The path to the document file.
pageNumber	The page number (starting at 0).
textOption	This parameter determines whether text elements are rendered as paths (which provides more accurate results) or as text.
includeAnnotations	Whether annotations (e.g., signatures) should be included in the imported document.

Returns

A Page containing a representation of the specified page from the document.

Definition at line 165 of file PDFParser.cs.

7.138.2.3 FromFile() [3/4]

```
static Document VectSharp.MuPDFUtils.PDFParser.FromFile (
            string fileName,
```

```
out Dictionary< string, string > linkDestinations,
SVGCreationOptions.TextOption textOption = SVGCreationOptions.TextOption.TextAsPath,
bool includeAnnotations = true )  [static]
```

Import all pages from a document.

Parameters

fileName	The path to the document file.
linkDestinations	When this method returns, this variable will contain a dictionary used to associate graphic action tags to hyperlinks. This can be used to enable such links when rendering the Page to a file.
textOption	This parameter determines whether text elements are rendered as paths (which provides more accurate results) or as text.
includeAnnotations	Whether annotations (e.g., signatures) should be included in the imported document.

Returns

A Document containing a representation of every page from the document.

Definition at line 282 of file PDFParser.cs.

7.138.2.4 FromFile() [4/4]

```
static Document VectSharp.MuPDFUtils.PDFParser.FromFile (
            string fileName,
            SVGCreationOptions.TextOption textOption = SVGCreationOptions.TextOption.TextAsPath,
            bool includeAnnotations = true )  [static]
```

Import all pages from a document.

Parameters

fileName	The path to the document file.
textOption	This parameter determines whether text elements are rendered as paths (which provides more accurate results) or as text.
includeAnnotations	Whether annotations (e.g., signatures) should be included in the imported document.

Returns

A Document containing a representation of every page from the document.

Definition at line 256 of file PDFParser.cs.

7.138.2.5 FromStream() [1/4]

```
static Document VectSharp.MuPDFUtils.PDFParser.FromStream (
            Stream stream,
            InputFileTypes fileType = InputFileTypes.PDF,
            SVGCreationOptions.TextOption textOption = SVGCreationOptions.TextOption.TextAsPath,
            bool includeAnnotations = true )  [static]
```

Import all pages from a document.

Parameters

stream	The Stream containing the document to import.
fileType	The type of document contained in the Stream.
textOption	This parameter determines whether text elements are rendered as paths (which provides more accurate results) or as text.
includeAnnotations	Whether annotations (e.g., signatures) should be included in the imported document.

Returns

A Document containing a representation of every page from the document.

Definition at line 312 of file PDFParser.cs.

7.138.2.6 FromStream() [2/4]

```
static Page VectSharp.MuPDFUtils.PDFParser.FromStream (
            Stream stream,
            int pageNumber,
            InputFileTypes fileType = InputFileTypes.PDF,
            SVGCreationOptions.TextOption textOption = SVGCreationOptions.TextOption.TextAsPath,
            bool includeAnnotations = true )  [static]
```

Import the specified page from a document.

Parameters

stream	The Stream containing the document to import.
pageNumber	The page number (starting at 0).
fileType	The type of document contained in the Stream.
textOption	This parameter determines whether text elements are rendered as paths (which provides more accurate results) or as text.
includeAnnotations	Whether annotations (e.g., signatures) should be included in the imported document.

Returns

A Page containing a representation of the specified page from the document.

Definition at line 208 of file PDFParser.cs.

7.138.2.7 FromStream() [3/4]

```
static Page VectSharp.MuPDFUtils.PDFParser.FromStream (
            Stream stream,
            int pageNumber,
            out Dictionary< string, string > linkDestinations,
            InputFileTypes fileType = InputFileTypes.PDF,
            SVGCreationOptions.TextOption textOption = SVGCreationOptions.TextOption.TextAsPath,
            bool includeAnnotations = true )  [static]
```

Import the specified page from a document.

Parameters

stream	The Stream containing the document to import.
pageNumber	The page number (starting at 0).
linkDestinations	When this method returns, this variable will contain a dictionary used to associate graphic action tags to hyperlinks. This can be used to enable such links when rendering the Page to a file.
fileType	The type of document contained in the Stream.
textOption	This parameter determines whether text elements are rendered as paths (which provides more accurate results) or as text.
includeAnnotations	Whether annotations (e.g., signatures) should be included in the imported document.

Returns

A Page containing a representation of the specified page from the document.

Definition at line 232 of file PDFParser.cs.

7.138.2.8 FromStream() [4/4]

```
static Document VectSharp.MuPDFUtils.PDFParser.FromStream (
            Stream stream,
            out Dictionary< string, string > linkDestinations,
            InputFileTypes fileType = InputFileTypes.PDF,
            SVGCreationOptions.TextOption textOption = SVGCreationOptions.TextOption.TextAsPath,
            bool includeAnnotations = true )  [static]
```

Import all pages from a document.

Parameters

stream	The Stream containing the document to import.
linkDestinations	When this method returns, this variable will contain a dictionary used to associate graphic action tags to hyperlinks. This can be used to enable such links when rendering the Page to a file.
fileType	The type of document contained in the Stream.
textOption	This parameter determines whether text elements are rendered as paths (which provides more accurate results) or as text.
includeAnnotations	Whether annotations (e.g., signatures) should be included in the imported document.

Returns

A Document containing a representation of every page from the document.

Definition at line 342 of file PDFParser.cs.

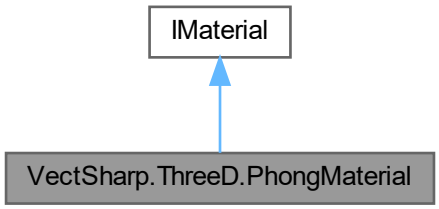
The documentation for this class was generated from the following file:

- VectSharp.MuPDFUtils/PDFParser.cs

7.139 VectSharp.ThreeD.PhongMaterial Class Reference

Represents a material that uses a Phong reflection model to determine the colour of the material based on the light sources that hit it.

Inheritance diagram for VectSharp.ThreeD.PhongMaterial:



Public Member Functions

- PhongMaterial (Colour colour)

  Creates a new PhongMaterial instance.
- Colour GetColour (Point3D point, NormalizedVector3D surfaceNormal, Camera camera, IList< ILightSource > lights, IList< double > obstructions)

  Obtains the Colour at the specified point.

Parameters

point	*The point whose colour should be determined.*
surfaceNormal	*The normal to the surface at the specified point .*
camera	*The camera being used to render the scene.*
lights	*A list of light sources that are present in the scene.*
obstructions	*A list of values indicating how obstructed each light source is.*

Returns

The Colour of the specified point.

Properties

- Colour Colour `[get]`

  The base colour of the material.
- double AmbientReflectionCoefficient = 1 `[get, set]`

  A coefficient determining how much ambient light is reflected by the material.
- double DiffuseReflectionCoefficient = 1 `[get, set]`

  A coefficient determining how much directional light is reflected by the material.
- double SpecularReflectionCoefficient = 1 `[get, set]`

  A coefficient determining the intensity of specular highlights.
- double SpecularShininess = 1 `[get, set]`

  A coefficient determining the extent of specular highlights.

7.139.1 Detailed Description

Represents a material that uses a Phong reflection model to determine the colour of the material based on the light sources that hit it.

Definition at line 74 of file Materials.cs.

7.139.2 Constructor & Destructor Documentation

7.139.2.1 PhongMaterial()

```
VectSharp.ThreeD.PhongMaterial.PhongMaterial (
            Colour colour )
```

Creates a new PhongMaterial instance.

Parameters

colour	The base colour of the material.

Definition at line 111 of file Materials.cs.

7.139.3 Member Function Documentation

7.139.3.1 GetColour()

```
Colour VectSharp.ThreeD.PhongMaterial.GetColour (
            Point3D point,
            NormalizedVector3D surfaceNormal,
            Camera camera,
            IList< ILightSource > lights,
            IList< double > obstructions )
```

Obtains the Colour at the specified point.

Parameters

point	The point whose colour should be determined.
surfaceNormal	The normal to the surface at the specified *point* .
camera	The camera being used to render the scene.
lights	A list of light sources that are present in the scene.
obstructions	A list of values indicating how obstructed each light source is.

Returns

The Colour of the specified point.

Implements VectSharp.ThreeD.IMaterial.

Definition at line 142 of file Materials.cs.

7.139.4 Property Documentation

7.139.4.1 AmbientReflectionCoefficient

```
double VectSharp.ThreeD.PhongMaterial.AmbientReflectionCoefficient = 1  [get], [set]
```

A coefficient determining how much ambient light is reflected by the material.

Definition at line 90 of file Materials.cs.

7.139.4.2 Colour

```
Colour VectSharp.ThreeD.PhongMaterial.Colour  [get]
```

The base colour of the material.

Definition at line 79 of file Materials.cs.

7.139.4.3 DiffuseReflectionCoefficient

```
double VectSharp.ThreeD.PhongMaterial.DiffuseReflectionCoefficient = 1  [get], [set]
```

A coefficient determining how much directional light is reflected by the material.

Definition at line 95 of file Materials.cs.

7.139.4.4 SpecularReflectionCoefficient

```
double VectSharp.ThreeD.PhongMaterial.SpecularReflectionCoefficient = 1  [get], [set]
```

A coefficient determining the intensity of specular highlights.

Definition at line 100 of file Materials.cs.

7.139.4.5 SpecularShininess

```
double VectSharp.ThreeD.PhongMaterial.SpecularShininess = 1  [get], [set]
```

A coefficient determining the extent of specular highlights.

Definition at line 105 of file Materials.cs.

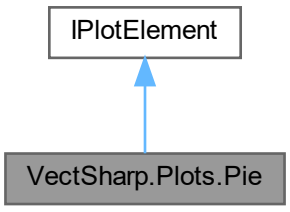
The documentation for this class was generated from the following file:

- VectSharp.ThreeD/Materials.cs

7.140 VectSharp.Plots.Pie Class Reference

A plot element that draws a pie or a doughnut.

Inheritance diagram for VectSharp.Plots.Pie:



Public Member Functions

- Pie (IReadOnlyList< double > data, IReadOnlyList< double > centre, IReadOnlyList< double > radius, ICoordinateSystem< IReadOnlyList< double > > coordinateSystem)

  Create a new Pie instance drawing a pie chart.
- Pie (IReadOnlyList< double > data, IReadOnlyList< double > centre, IReadOnlyList< double > innerRadius, IReadOnlyList< double > outerRadius, ICoordinateSystem< IReadOnlyList< double > > coordinateSystem)

  Create a new Pie instance drawing a doughnut chart.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IReadOnlyList< double > Data `[get, set]`

  The data in the pie chart. The values do not need to be normalised.
- IReadOnlyList< double > Centre `[get, set]`

  The centre of the pie/doughnut, in data coordinates.
- IReadOnlyList< double > OuterRadius `[get, set]`

  The outer radius of the pie/doughnut, in data coordinates.
- IReadOnlyList< double > InnerRadius `[get, set]`

  The inner radius of the doughnut, in data coordinates. Set to [0, 0] for a pie chart.
- double StartAngle = 0 `[get, set]`

  The initial angle starting from which the pie slices are drawn.
- bool Clockwise = false `[get, set]`

  Determines whether the slices are drawn in clockwise or anti-clockwise fashion.
- ICoordinateSystem< IReadOnlyList< double > > CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.
- IReadOnlyList< PlotElementPresentationAttributes > PresentationAttributes = new PlotElementPresentationAttributes[ ] { new PlotElementPresentationAttributes() } `[get, set]`

  Presentation attributes for the slices. An element from this collection is used for each slice in the pie/doughnut; if there are more slices than elements in this collection, the presentation attributes are wrapped.
- string Tag `[get, set]`

  A tag to identify the pie/doughnut in the plot.

7.140.1 Detailed Description

A plot element that draws a pie or a doughnut.

Definition at line 27 of file Pie.cs.

7.140.2 Constructor & Destructor Documentation

7.140.2.1 Pie() [1/2]

```
VectSharp.Plots.Pie.Pie (
            IReadOnlyList< double > data,
            IReadOnlyList< double > centre,
            IReadOnlyList< double > radius,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem )
```

Create a new Pie instance drawing a pie chart.

Parameters

data	The data in the pie chart. The values do not need to be normalised.
centre	The centre of the pie, in data coordinates.
radius	The radius of the pie, in data coordinates.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 84 of file Pie.cs.

7.140.2.2 Pie() [2/2]

```
VectSharp.Plots.Pie.Pie (
            IReadOnlyList< double > data,
            IReadOnlyList< double > centre,
            IReadOnlyList< double > innerRadius,
            IReadOnlyList< double > outerRadius,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem )
```

Create a new Pie instance drawing a doughnut chart.

Parameters

data	The data in the doughnut chart. The values do not need to be normalised.
centre	The centre of the doughnut, in data coordinates.
innerRadius	The inner radius of the doughnut, in data coordinates.
outerRadius	The outer radius of the doughnut, in data coordinates.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 101 of file Pie.cs.

7.140.3 Member Function Documentation

7.140.3.1 Plot()

```
void VectSharp.Plots.Pie.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 111 of file Pie.cs.

7.140.4 Property Documentation

7.140.4.1 Centre

```
IReadOnlyList<double> VectSharp.Plots.Pie.Centre  [get], [set]
```

The centre of the pie/doughnut, in data coordinates.

Definition at line 37 of file Pie.cs.

7.140.4.2 Clockwise

```
bool VectSharp.Plots.Pie.Clockwise = false  [get], [set]
```

Determines whether the slices are drawn in clockwise or anti-clockwise fashion.

Definition at line 57 of file Pie.cs.

7.140.4.3 CoordinateSystem

```
ICoordinateSystem<IReadOnlyList<double> > VectSharp.Plots.Pie.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 62 of file Pie.cs.

7.140.4.4 Data

```
IReadOnlyList<double> VectSharp.Plots.Pie.Data  [get], [set]
```

The data in the pie chart. The values do not need to be normalised.

Definition at line 32 of file Pie.cs.

7.140.4.5 InnerRadius

```
IReadOnlyList<double> VectSharp.Plots.Pie.InnerRadius  [get], [set]
```

The inner radius of the doughnut, in data coordinates. Set to [0, 0] for a pie chart.

Definition at line 47 of file Pie.cs.

7.140.4.6 OuterRadius

```
IReadOnlyList<double> VectSharp.Plots.Pie.OuterRadius  [get], [set]
```

The outer radius of the pie/doughnut, in data coordinates.

Definition at line 42 of file Pie.cs.

7.140.4.7 PresentationAttributes

```
IReadOnlyList<PlotElementPresentationAttributes> VectSharp.Plots.Pie.PresentationAttributes
= new PlotElementPresentationAttributes[] { new PlotElementPresentationAttributes() }  [get],
[set]
```

Presentation attributes for the slices. An element from this collection is used for each slice in the pie/doughnut; if there are more slices than elements in this collection, the presentation attributes are wrapped.

Definition at line 70 of file Pie.cs.

7.140.4.8 StartAngle

```
double VectSharp.Plots.Pie.StartAngle = 0  [get], [set]
```

The initial angle starting from which the pie slices are drawn.

Definition at line 52 of file Pie.cs.

7.140.4.9 Tag

```
string VectSharp.Plots.Pie.Tag  [get], [set]
```

A tag to identify the pie/doughnut in the plot.

Definition at line 75 of file Pie.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/Pie.cs

7.141 VectSharp.Plots.Plot Class Reference

Represents a collection of plot elements.

Public Types

- enum WhiskerType

  Describes the types of whiskers for a box plot.
- enum NormalisationMode

  Describes the kind of normalisation that is performed.

Public Member Functions

- void AddPlotElement (IPlotElement plotElement)

  Add the specified plot element to the plot.
- void AddPlotElements (IEnumerable< IPlotElement > plotElements)

  Add the specified plot elements to the plot.
- void AddPlotElements (params IPlotElement[ ] plotElements)

  Add the specified plot elements to the plot.
- void RemovePlotElement (IPlotElement plotElement)

  Remove the specified plot element from the plot.
- void Render (Graphics target)

  Render the plot on the specified target Graphics.
- T GetFirst< T > ()

  Get the first IPlotElement of the specified type, or the first ICoordinateSystem of the specified type.
- IEnumerable< T > GetAll< T > ()

  Get all IPlotElements of the specified type, or all ICoordinateSystems of the specified type.
- Page Render ()

  Render the plot to a suitably cropped Page object.
- Plot ()

  Create a new empty Plot.

Properties

- ImmutableList< IPlotElement > PlotElements = ImmutableList.Create<IPlotElement>() `[get]`

  The elements contained in the plot.

7.141.1 Detailed Description

Represents a collection of plot elements.

Definition at line 24 of file Plot.Area.cs.

7.141.2 Member Enumeration Documentation

7.141.2.1 NormalisationMode

```
enum VectSharp.Plots.Plot.NormalisationMode
```

Describes the kind of normalisation that is performed.

Definition at line 29 of file Plot.Histogram.cs.

7.141.2.2 WhiskerType

```
enum VectSharp.Plots.Plot.WhiskerType
```

Describes the types of whiskers for a box plot.

Definition at line 30 of file Plot.BoxPlot.cs.

7.141.3 Constructor & Destructor Documentation

7.141.3.1 Plot()

```
VectSharp.Plots.Plot.Plot ( )
```

Create a new empty Plot.

Definition at line 260 of file Plot.cs.

7.141.4 Member Function Documentation

7.141.4.1 AddPlotElement()

```
void VectSharp.Plots.Plot.AddPlotElement (
            IPlotElement plotElement )
```

Add the specified plot element to the plot.

Parameters

plotElement	The plot element to add.

Definition at line 154 of file Plot.cs.

7.141.4.2 AddPlotElements() [1/2]

```
void VectSharp.Plots.Plot.AddPlotElements (
            IEnumerable< IPlotElement > plotElements )
```

Add the specified plot elements to the plot.

Parameters

plotElements	The plot elements to add.

Definition at line 163 of file Plot.cs.

7.141.4.3 AddPlotElements() [2/2]

```
void VectSharp.Plots.Plot.AddPlotElements (
            params IPlotElement[] plotElements )
```

Add the specified plot elements to the plot.

Parameters

plotElements	The plot elements to add.

Definition at line 172 of file Plot.cs.

7.141.4.4 GetAll< T >()

```
IEnumerable< T > VectSharp.Plots.Plot.GetAll< T > ( )
```

Get all IPlotElements of the specified type, or all ICoordinateSystems of the specified type.

Template Parameters

T	The type of plot element or coordinate system to get.

Returns

All IPlotElements of the specified type, or all ICoordinateSystems of the specified type.

Type Constraints

T* : *class

T* : *IPlotElement

Definition at line 225 of file Plot.cs.

7.141.4.5 GetFirst< T >()

```
T VectSharp.Plots.Plot.GetFirst< T > ( )
```

Get the first IPlotElement of the specified type, or the first ICoordinateSystem of the specified type.

Template Parameters

T	The type of plot element or coordinate system to get.

Returns

The first IPlotElement of the specified type, or the first ICoordinateSystem of the specified type, or `null` if none could be found.

Type Constraints

T* : *class

Definition at line 203 of file Plot.cs.

7.141.4.6 RemovePlotElement()

```
void VectSharp.Plots.Plot.RemovePlotElement (
            IPlotElement plotElement )
```

Remove the specified plot element from the plot.

Parameters

plotElement	The plot element to remove.

Definition at line 181 of file Plot.cs.

7.141.4.7 Render() [1/2]

```
Page VectSharp.Plots.Plot.Render ( )
```

Render the plot to a suitably cropped Page object.

Returns

A Page object containing the rendered plot.

Definition at line 244 of file Plot.cs.

7.141.4.8 Render() [2/2]

```
void VectSharp.Plots.Plot.Render (
            Graphics target )
```

Render the plot on the specified *target* Graphics.

Parameters

target	The Graphics on which the plot should be drawn.

Definition at line 190 of file Plot.cs.

7.141.5 Property Documentation

7.141.5.1 PlotElements

```
ImmutableList<IPlotElement> VectSharp.Plots.Plot.PlotElements = ImmutableList.Create<IPlotElement>()
[get]
```

The elements contained in the plot.

Definition at line 148 of file Plot.cs.

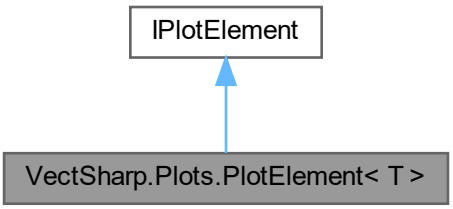
The documentation for this class was generated from the following files:

- VectSharp.Plots/Plot.Area.cs
- VectSharp.Plots/Plot.cs
- VectSharp.Plots/Plot.Histogram.cs

7.142 VectSharp.Plots.PlotElement< T > Class Template Reference

A plot element that uses an Action to draw its contents.

Inheritance diagram for VectSharp.Plots.PlotElement< T >:



Public Member Functions

- PlotElement (ICoordinateSystem< T > coordinateSystem, Action< Graphics, ICoordinateSystem< T > > plotAction)

  Create a new PlotElement< T > using the specified coordinate system and plot action.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- Action< Graphics, ICoordinateSystem< T > > PlotAction `[get, set]`

  The action that is invoked when the plot element needs to be drawn.
- ICoordinateSystem< T > CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.

7.142.1 Detailed Description

A plot element that uses an Action to draw its contents.

Template Parameters

T	The type of data to plot (e.g. `double[]`).

Definition at line 109 of file Plot.cs.

7.142.2 Constructor & Destructor Documentation

7.142.2.1 PlotElement()

```
VectSharp.Plots.PlotElement< T >.PlotElement (
            ICoordinateSystem< T > coordinateSystem,
            Action< Graphics, ICoordinateSystem< T > > plotAction )
```

Create a new PlotElement<T> using the specified coordinate system and plot action.

Parameters

coordinateSystem	The coordinate system used to transform the points from data space to plot space.
plotAction	The action that is invoked when the plot element needs to be drawn.

Definition at line 127 of file Plot.cs.

7.142.3 Member Function Documentation

7.142.3.1 Plot()

```
void VectSharp.Plots.PlotElement< T >.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 134 of file Plot.cs.

7.142.4 Property Documentation

7.142.4.1 CoordinateSystem

```
ICoordinateSystem<T> VectSharp.Plots.PlotElement< T >.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 119 of file Plot.cs.

7.142.4.2 PlotAction

```
Action<Graphics, ICoordinateSystem<T> > VectSharp.Plots.PlotElement< T >.PlotAction [get], [set]
```

The action that is invoked when the plot element needs to be drawn.

Definition at line 114 of file Plot.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/Plot.cs

7.143 VectSharp.Plots.PlotElementPresentationAttributes Class Reference

Determines the appearance of plot elements.

Public Member Functions

- PlotElementPresentationAttributes ()
  Create a new PlotElementPresentationAttributes with the default values.
- PlotElementPresentationAttributes (PlotElementPresentationAttributes other)
  Creates a new PlotElementPresentationAttributes copying all settings from another instance.

Properties

- Brush Stroke = Colours.Black [get, set]
  The stroke of the plot element.
- Brush Fill = Colours.Black [get, set]
  The fill of the plot element.
- double LineWidth = 1 [get, set]
  The thickness of lines in the plot element.
- LineDash? LineDash = null [get, set]
  The line dash style for the plot element.
- LineCaps LineCap = LineCaps.Square [get, set]
  The line cap for the plot element.
- LineJoins LineJoin = LineJoins.Miter [get, set]
  The line join for the plot element.
- Font Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica), 12) [get, set]
  The font used to draw text in the plot element.

7.143.1 Detailed Description

Determines the appearance of plot elements.

Definition at line 27 of file Plot.cs.

7.143.2 Constructor & Destructor Documentation

7.143.2.1 PlotElementPresentationAttributes() [1/2]

```
VectSharp.Plots.PlotElementPresentationAttributes.PlotElementPresentationAttributes ( )
```

Create a new PlotElementPresentationAttributes with the default values.

Definition at line 67 of file Plot.cs.

7.143.2.2 PlotElementPresentationAttributes() [2/2]

```
VectSharp.Plots.PlotElementPresentationAttributes.PlotElementPresentationAttributes (
            PlotElementPresentationAttributes other )
```

Creates a new PlotElementPresentationAttributes copying all settings from another instance.

Parameters

other	The other instance from which the settings will be copied.

Definition at line 76 of file Plot.cs.

7.143.3 Property Documentation

7.143.3.1 Fill

```
Brush VectSharp.Plots.PlotElementPresentationAttributes.Fill = Colours.Black  [get], [set]
```

The fill of the plot element.

Definition at line 37 of file Plot.cs.

7.143.3.2 Font

```
Font VectSharp.Plots.PlotElementPresentationAttributes.Font = new Font(FontFamily.ResolveFontFamily(Font↩
Family.StandardFontFamilies.Helvetica), 12)  [get], [set]
```

The font used to draw text in the plot element.

Definition at line 62 of file Plot.cs.

7.143.3.3 LineCap

```
LineCaps VectSharp.Plots.PlotElementPresentationAttributes.LineCap = LineCaps.Square  [get],
[set]
```

The line cap for the plot element.

Definition at line 52 of file Plot.cs.

7.143.3.4 LineDash

```
LineDash?  VectSharp.Plots.PlotElementPresentationAttributes.LineDash = null  [get], [set]
```

The line dash style for the plot element.

Definition at line 47 of file Plot.cs.

7.143.3.5 LineJoin

```
LineJoins VectSharp.Plots.PlotElementPresentationAttributes.LineJoin = LineJoins.Miter  [get],
[set]
```

The line join for the plot element.

Definition at line 57 of file Plot.cs.

7.143.3.6 LineWidth

```
double VectSharp.Plots.PlotElementPresentationAttributes.LineWidth = 1  [get], [set]
```

The thickness of lines in the plot element.

Definition at line 42 of file Plot.cs.

7.143.3.7 Stroke

```
Brush VectSharp.Plots.PlotElementPresentationAttributes.Stroke = Colours.Black  [get], [set]
```

The stroke of the plot element.

Definition at line 32 of file Plot.cs.

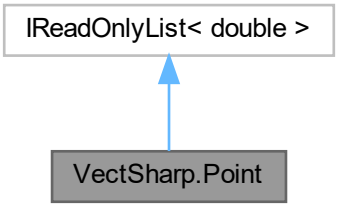
The documentation for this class was generated from the following file:

- VectSharp.Plots/Plot.cs

7.144 VectSharp.Point Struct Reference

Represents a point relative to an origin in the top-left corner.

Inheritance diagram for VectSharp.Point:



Public Member Functions

- Point (double x, double y)

  Create a new Point.
- double Modulus ()

  Computes the modulus of the vector represented by the Point.
- Point Normalize ()

  Normalises a Point.
- bool IsEqual (Point p2, double tolerance)

  Checks whether this Point is equal to another Point, up to a specified tolerance.
- IEnumerator< double > GetEnumerator ()
- IEnumerator IEnumerable. **GetEnumerator** ()

Static Public Member Functions

- static Point Min (Point p1, Point p2)

  Computes the top-left corner of the Rectangle identified by two Points.
- static Point Max (Point p1, Point p2)

  Computes the bottom-right corner of the Rectangle identified by two Points.
- static Rectangle Bounds (IEnumerable< Point > points)

  Computes the smallest Rectangle that contains all the specified points.
- static Rectangle Bounds (params Point[ ] points)

  Computes the smallest Rectangle that contains all the specified points.
- static implicit operator (double X, double Y)(Point point)

  Implicit conversion to a tuple.
- static implicit operator double[ ] (Point point)

  Implicit conversion to an array.
- static implicit operator Point ((double X, double Y) tuple)

  Implicit conversion from a tuple.
- static operator Point (double[ ] array)

  Explicit conversion from an array (will throw an exception if the array does not contain exactly two elements).
- static Point operator- (Point p1, Point p2)

  Subtract the coordinates of two points.
- static Point operator+ (Point p1, Point p2)

  Add the coordinates of two points.
- static Point operator* (double t, Point p)

  Multiply the coordinates of a point by a scalar.
- static Point operator* (Point p, double t)

  Multiply the coordinates of a point by a scalar.

Public Attributes

- double X

  Horizontal (x) coordinate, measured to the right of the origin.
- double Y

  Vertical (y) coordinate, measured to the bottom of the origin.

Properties

- int Count `[get]`
- double this[int index] `[get]`

7.144.1 Detailed Description

Represents a point relative to an origin in the top-left corner.

Definition at line 31 of file Point.cs.

7.144.2 Constructor & Destructor Documentation

7.144.2.1 Point()

```
VectSharp.Point.Point (
            double x,
            double y )
```

Create a new Point.

Parameters

x	The horizontal (x) coordinate.
y	The vertical (y) coordinate.

Definition at line 71 of file Point.cs.

7.144.3 Member Function Documentation

7.144.3.1 Bounds() [1/2]

```
static Rectangle VectSharp.Point.Bounds (
            IEnumerable< Point > points )  [static]
```

Computes the smallest Rectangle that contains all the specified points.

Parameters

points	The points whose bounds are being computed.

Returns

The smallest Rectangle that contains all the specified points.

Definition at line 135 of file Point.cs.

7.144.3.2 Bounds() [2/2]

```
static Rectangle VectSharp.Point.Bounds (
            params Point[] points )  [static]
```

Computes the smallest Rectangle that contains all the specified points.

Parameters

points	The points whose bounds are being computed.

Returns

The smallest Rectangle that contains all the specified points.

Definition at line 164 of file Point.cs.

7.144.3.3 GetEnumerator()

```
IEnumerator< double > VectSharp.Point.GetEnumerator ( )
```

Definition at line 170 of file Point.cs.

7.144.3.4 IsEqual()

```
bool VectSharp.Point.IsEqual (
            Point p2,
            double tolerance )
```

Checks whether this Point is equal to another Point, up to a specified tolerance.

Parameters

p2	The Point to compare.
tolerance	The tolerance threshold.

Returns

`true` if both coordinates of the Points are closer than *tolerance* or if their relative difference (i.e. `(a - b) / (a + b) * 2`) is smaller than *tolerance* . `false` otherwise.

Definition at line 103 of file Point.cs.

7.144.3.5 Max()

```
static Point VectSharp.Point.Max (
            Point p1,
            Point p2 )  [static]
```

Computes the bottom-right corner of the Rectangle identified by two Points.

Parameters

p1	The first point.
p2	The second point.

Returns

A Point whose X coordinate is the largest between the one of *p1* and *p2* , and likewise for the Y coordinate.

Definition at line 125 of file Point.cs.

7.144.3.6 Min()

```
static Point VectSharp.Point.Min (
            Point p1,
            Point p2 )  [static]
```

Computes the top-left corner of the Rectangle identified by two Points.

Parameters

p1	The first point.
p2	The second point.

Returns

A Point whose X coordinate is the smallest between the one of *p1* and *p2* , and likewise for the Y coordinate.

Definition at line 114 of file Point.cs.

7.144.3.7 Modulus()

```
double VectSharp.Point.Modulus ( )
```

Computes the modulus of the vector represented by the Point.

Returns

The modulus of the vector represented by the Point.

Definition at line 81 of file Point.cs.

7.144.3.8 Normalize()

```
Point VectSharp.Point.Normalize ( )
```

Normalises a Point.

Returns

The normalised Point.

Definition at line 91 of file Point.cs.

7.144.3.9 operator()

```
static implicit VectSharp.Point.operator (
            double X,
            double Y )  [static]
```

Implicit conversion to a tuple.

Parameters

point	The point to convert.

Definition at line 183 of file Point.cs.

7.144.3.10 operator double[]()

```
static implicit VectSharp.Point.operator double[] (
            Point point )  [static]
```

Implicit conversion to an array.

Parameters

point	The point to convert

Definition at line 192 of file Point.cs.

7.144.3.11 operator Point() [1/2]

```
static implicit VectSharp.Point.operator Point (
            (double X, double Y) tuple )  [static]
```

Implicit conversion from a tuple.

Parameters

tuple	The tuple to convert.

Definition at line 201 of file Point.cs.

7.144.3.12 operator Point() [2/2]

```
static VectSharp.Point.operator Point (
            double[] array )  [explicit], [static]
```

Explicit conversion from an array (will throw an exception if the array does not contain exactly two elements).

Parameters

array	The array to convert. It must contain exactly two elements.

Definition at line 210 of file Point.cs.

7.144.3.13 operator∗() [1/2]

```
static Point VectSharp.Point.operator* (
            double t,
            Point p )  [static]
```

Multiply the coordinates of a point by a scalar.

Parameters

t	The scalar.
p	The point.

Returns

A new Point whose coordinates are the product of the original points' and the scalar.

Definition at line 252 of file Point.cs.

7.144.3.14 operator∗() [2/2]

```
static Point VectSharp.Point.operator* (
            Point p,
            double t )  [static]
```

Multiply the coordinates of a point by a scalar.

Parameters

t	The scalar.
p	The point.

Returns

A new Point whose coordinates are the product of the original points' and the scalar.

Definition at line 263 of file Point.cs.

7.144.3.15 operator+()

```
static Point VectSharp.Point.operator+ (
            Point p1,
            Point p2 )  [static]
```

Add the coordinates of two points.

Parameters

p1	The first point.
p2	The second point.

Returns

A new Point whose coordinates are the sum of the coordinates of the original points.

Definition at line 241 of file Point.cs.

7.144.3.16 operator-()

```
static Point VectSharp.Point.operator- (
            Point p1,
            Point p2 )  [static]
```

Subtract the coordinates of two points.

Parameters

p1	The first point.
p2	The second point.

Returns

A new Point whose coordinates are the difference between the coordinates of the original points.

Definition at line 230 of file Point.cs.

7.144.4 Member Data Documentation

7.144.4.1 X

```
double VectSharp.Point.X
```

Horizontal (x) coordinate, measured to the right of the origin.

Definition at line 36 of file Point.cs.

7.144.4.2 Y

```
double VectSharp.Point.Y
```

Vertical (y) coordinate, measured to the bottom of the origin.

Definition at line 41 of file Point.cs.

7.144.5 Property Documentation

7.144.5.1 Count

```
int VectSharp.Point.Count  [get]
```

Definition at line 44 of file Point.cs.

7.144.5.2 this[int index]

```
double VectSharp.Point.this[int index]  [get]
```

Definition at line 47 of file Point.cs.

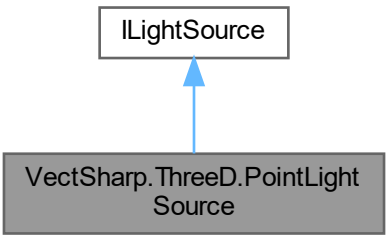
The documentation for this struct was generated from the following file:

- VectSharp/Point.cs

7.145 VectSharp.ThreeD.PointLightSource Class Reference

Represents a point light source.

Inheritance diagram for VectSharp.ThreeD.PointLightSource:



Public Member Functions

- PointLightSource (double intensity, Point3D position)

  Creates a new PointLightSource instance.
- LightIntensity GetLightAt (Point3D point)

  Computes the light intensity at the specified point, without taking into account any obstructions.

Parameters

point	*The Point3DElement at which the light intensity should be computed.*

Returns

- double GetObstruction (Point3D point, IEnumerable< Triangle3DElement > shadowingTriangles)

  Determines the amount of obstruction of the light that results at point due to the specified shadowingTriangles .

Parameters

point	*The Point3D at which the obstruction should be computed.*
shadowingTriangles	*A collection of Triangle3DElement casting shadows.*

Returns

1 if the light is completely obstructed, 0 if the light is completely visible, a value between these if the light is partially obstructed.

Properties

- bool CastsShadow = true `[get, set]`

  Determines whether the light casts a shadow or not.
- Point3D Position `[get, set]`

  The position of the light source.
- double Intensity `[get, set]`

  The base intensity of the light.
- double DistanceAttenuationExponent = 2 `[get, set]`

  An exponent determining how fast the light attenuates with increasing distance. Set to 0 to disable distance attenuation.

7.145.1 Detailed Description

Represents a point light source.

Definition at line 184 of file Lights.cs.

7.145.2 Constructor & Destructor Documentation

7.145.2.1 PointLightSource()

```
VectSharp.ThreeD.PointLightSource.PointLightSource (
            double intensity,
            Point3D position )
```

Creates a new PointLightSource instance.

Parameters

intensity	The intensity of the light.
position	The position of the light source.

Definition at line 209 of file Lights.cs.

7.145.3 Member Function Documentation

7.145.3.1 GetLightAt()

```
LightIntensity VectSharp.ThreeD.PointLightSource.GetLightAt (
            Point3D point )
```

Computes the light intensity at the specified point, without taking into account any obstructions.

Parameters

point	The Point3DElement at which the light intensity should be computed.

Returns

Implements VectSharp.ThreeD.ILightSource.

Definition at line 216 of file Lights.cs.

7.145.3.2 GetObstruction()

```
double VectSharp.ThreeD.PointLightSource.GetObstruction (
            Point3D point,
            IEnumerable< Triangle3DElement > shadowingTriangles )
```

Determines the amount of obstruction of the light that results at *point* due to the specified *shadowingTriangles* .

Parameters

point	The Point3D at which the obstruction should be computed.
shadowingTriangles	A collection of Triangle3DElement casting shadows.

Returns

1 if the light is completely obstructed, 0 if the light is completely visible, a value between these if the light is partially obstructed.

Implements VectSharp.ThreeD.ILightSource.

Definition at line 233 of file Lights.cs.

7.145.4 Property Documentation

7.145.4.1 CastsShadow

```
bool VectSharp.ThreeD.PointLightSource.CastsShadow = true  [get], [set]
```

Determines whether the light casts a shadow or not.

Implements VectSharp.ThreeD.ILightSource.

Definition at line 187 of file Lights.cs.

7.145.4.2 DistanceAttenuationExponent

```
double VectSharp.ThreeD.PointLightSource.DistanceAttenuationExponent = 2  [get], [set]
```

An exponent determining how fast the light attenuates with increasing distance. Set to 0 to disable distance attenuation.

Definition at line 202 of file Lights.cs.

7.145.4.3 Intensity

```
double VectSharp.ThreeD.PointLightSource.Intensity  [get], [set]
```

The base intensity of the light.

Definition at line 197 of file Lights.cs.

7.145.4.4 Position

```
Point3D VectSharp.ThreeD.PointLightSource.Position  [get], [set]
```

The position of the light source.

Definition at line 192 of file Lights.cs.

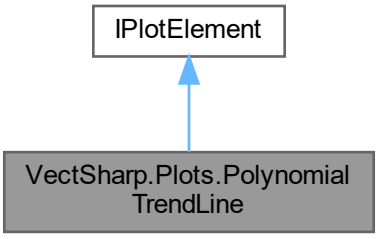
The documentation for this class was generated from the following file:

- VectSharp.ThreeD/Lights.cs

7.146 VectSharp.Plots.PolynomialTrendLine Class Reference

A plot element that draws a polynomial trendline with equation `y = a0 + a1 * x + a2 * x^2 + ... + aN * x^N`.

Inheritance diagram for VectSharp.Plots.PolynomialTrendLine:



Public Member Functions

- PolynomialTrendLine (double[] coefficients, double minX, double minY, double maxX, double maxY, IContinuousCoordinateSystem coordinateSystem)

  Create a new LinearTrendLine instance, specifying the coefficients.
- PolynomialTrendLine (IReadOnlyList< IReadOnlyList< double > > data, int order, IContinuousCoordinateSystem coordinateSystem, double? fixedIntercept=null)

  Create a new PolynomialTrendLine instance, determining the coefficients by running a regression.
- PolynomialTrendLine (IReadOnlyList<(double, double)> data, int order, IContinuousCoordinateSystem coordinateSystem, double? fixedIntercept=null)

  Create a new PolynomialTrendLine instance, determining the coefficients by running a regression.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- double[] Coefficients `[get, set]`

  The coefficients (`a0 ... aN`).
- double MinX `[get, set]`

  The minimum X value for which the trendline is plotted.
- double MinY `[get, set]`

  The minimum Y value for which the trendline is plotted.
- double MaxX `[get, set]`

  The maximum X value for which the trendline is plotted.
- double MaxY `[get, set]`

  The maximum Y value for which the trendline is plotted.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180, 180, 180) } `[get, set]`

  Presentation attributes for the trendline.
- string Tag `[get, set]`

  A tag to identify the trendline in the plot.
- IContinuousCoordinateSystem CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.

7.146.1 Detailed Description

A plot element that draws a polynomial trendline with equation `y = a0 + a1 * x + a2 * x^2 + ... + aN * x^N`.

Definition at line 685 of file Trendlines.cs.

7.146.2 Constructor & Destructor Documentation

7.146.2.1 PolynomialTrendLine() [1/3]

```
VectSharp.Plots.PolynomialTrendLine.PolynomialTrendLine (
            double[] coefficients,
            double minX,
            double minY,
            double maxX,
            double maxY,
            IContinuousCoordinateSystem coordinateSystem )
```

Create a new LinearTrendLine instance, specifying the coefficients.

Parameters

coefficients	The coefficients (`a0 ... aN`).
minX	The minimum X value for which the trendline is plotted.
minY	The minimum Y value for which the trendline is plotted.
maxX	The maximum X value for which the trendline is plotted.
maxY	The maximum Y value for which the trendline is plotted.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 736 of file Trendlines.cs.

7.146.2.2 PolynomialTrendLine() [2/3]

```
VectSharp.Plots.PolynomialTrendLine.PolynomialTrendLine (
            IReadOnlyList< IReadOnlyList< double > > data,
            int order,
            IContinuousCoordinateSystem coordinateSystem,
            double?  fixedIntercept = null )
```

Create a new PolynomialTrendLine instance, determining the coefficients by running a regression.

Parameters

data	The data that will be used to determine the coefficients.
order	The order of the polynomial (N). This must be ≥ 2.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.
fixedIntercept	If this is null, the intercept (a0) is determined during the regression; otherwise, it is fixed to the specified value.

Exceptions

ArgumentOutOfRangeException	Thrown if the *order* is < 2.

Definition at line 777 of file Trendlines.cs.

7.146.2.3 PolynomialTrendLine() [3/3]

```
VectSharp.Plots.PolynomialTrendLine.PolynomialTrendLine (
            IReadOnlyList<(double, double)> data,
            int order,
            IContinuousCoordinateSystem coordinateSystem,
            double?  fixedIntercept = null )
```

Create a new PolynomialTrendLine instance, determining the coefficients by running a regression.

Parameters

data	The data that will be used to determine the coefficients.
order	The order of the polynomial (N). This must be ≥ 2.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.
fixedIntercept	If this is null, the intercept (a0) is determined during the regression; otherwise, it is fixed to the specified value.

Exceptions

ArgumentOutOfRangeException	Thrown if the *order* is < 2.

Definition at line 830 of file Trendlines.cs.

7.146.3 Member Function Documentation

7.146.3.1 Plot()

```
void VectSharp.Plots.PolynomialTrendLine.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 833 of file Trendlines.cs.

7.146.4 Property Documentation

7.146.4.1 Coefficients

```
double [] VectSharp.Plots.PolynomialTrendLine.Coefficients  [get], [set]
```

The coefficients (`a0 ... aN`).

Definition at line 690 of file Trendlines.cs.

7.146.4.2 CoordinateSystem

```
IContinuousCoordinateSystem VectSharp.Plots.PolynomialTrendLine.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 724 of file Trendlines.cs.

7.146.4.3 MaxX

```
double VectSharp.Plots.PolynomialTrendLine.MaxX  [get], [set]
```

The maximum X value for which the trendline is plotted.

Definition at line 704 of file Trendlines.cs.

7.146.4.4 MaxY

```
double VectSharp.Plots.PolynomialTrendLine.MaxY  [get], [set]
```

The maximum Y value for which the trendline is plotted.

Definition at line 709 of file Trendlines.cs.

7.146.4.5 MinX

```
double VectSharp.Plots.PolynomialTrendLine.MinX  [get], [set]
```

The minimum X value for which the trendline is plotted.

Definition at line 694 of file Trendlines.cs.

7.146.4.6 MinY

```
double VectSharp.Plots.PolynomialTrendLine.MinY  [get], [set]
```

The minimum Y value for which the trendline is plotted.

Definition at line 699 of file Trendlines.cs.

7.146.4.7 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.PolynomialTrendLine.PresentationAttributes =
new PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180,
180, 180) }  [get], [set]
```

Presentation attributes for the trendline.

Definition at line 714 of file Trendlines.cs.

7.146.4.8 Tag

```
string VectSharp.Plots.PolynomialTrendLine.Tag  [get], [set]
```

A tag to identify the trendline in the plot.

Definition at line 719 of file Trendlines.cs.

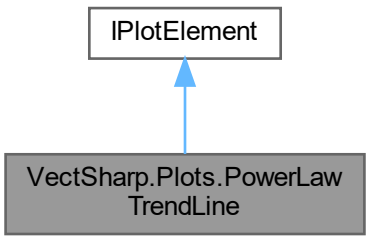
The documentation for this class was generated from the following file:

- VectSharp.Plots/Trendlines.cs

7.147 VectSharp.Plots.PowerLawTrendLine Class Reference

A plot element that draws a power law trendline with equation `y = b * x^a`.

Inheritance diagram for VectSharp.Plots.PowerLawTrendLine:



Public Member Functions

- PowerLawTrendLine (double slope, double intercept, double minX, double minY, double maxX, double maxY, IContinuousCoordinateSystem coordinateSystem)

  Create a new PowerLawTrendLine instance, specifying the equation parameters.
- PowerLawTrendLine (IReadOnlyList< IReadOnlyList< double > > data, IContinuousCoordinateSystem coordinateSystem)

  Create a new PowerLawTrendLine instance, determining the equation parameters by running a regression.
- PowerLawTrendLine (IReadOnlyList<(double, double)> data, IContinuousCoordinateSystem coordinate↩System)

  Create a new PowerLawTrendLine instance, determining the equation parameters by running a regression.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- double Slope `[get, set]`

  The slope of the trendline (a).
- double Intercept `[get, set]`

  The intercept of the trendline (b).
- double MinX `[get, set]`

  The minimum X value for which the trendline is plotted.
- double MinY `[get, set]`

  The minimum Y value for which the trendline is plotted.
- double MaxX `[get, set]`

  The maximum X value for which the trendline is plotted.
- double MaxY `[get, set]`

  The maximum Y value for which the trendline is plotted.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180, 180, 180) } `[get, set]`

  Presentation attributes for the trendline.
- string Tag `[get, set]`

  A tag to identify the trendline in the plot.
- IContinuousCoordinateSystem CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.

7.147.1 Detailed Description

A plot element that draws a power law trendline with equation `y = b * x^a`.

Definition at line 877 of file Trendlines.cs.

7.147.2 Constructor & Destructor Documentation

7.147.2.1 PowerLawTrendLine() [1/3]

```
VectSharp.Plots.PowerLawTrendLine.PowerLawTrendLine (
            double slope,
            double intercept,
            double minX,
            double minY,
            double maxX,
            double maxY,
            IContinuousCoordinateSystem coordinateSystem )
```

Create a new PowerLawTrendLine instance, specifying the equation parameters.

Parameters

slope	The slope of the trendline (a).
intercept	The intercept of the trendline (b).
minX	The minimum X value for which the trendline is plotted.
minY	The minimum Y value for which the trendline is plotted.
maxX	The maximum X value for which the trendline is plotted.
maxY	The maximum Y value for which the trendline is plotted.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 935 of file Trendlines.cs.

7.147.2.2 PowerLawTrendLine() [2/3]

```
VectSharp.Plots.PowerLawTrendLine.PowerLawTrendLine (
            IReadOnlyList< IReadOnlyList< double > > data,
            IContinuousCoordinateSystem coordinateSystem )
```

Create a new PowerLawTrendLine instance, determining the equation parameters by running a regression.

Parameters

data	The data that will be used to determine the equation parameters.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 951 of file Trendlines.cs.

7.147.2.3 PowerLawTrendLine() [3/3]

```
VectSharp.Plots.PowerLawTrendLine.PowerLawTrendLine (
            IReadOnlyList<(double, double)> data,
            IContinuousCoordinateSystem coordinateSystem )
```

Create a new PowerLawTrendLine instance, determining the equation parameters by running a regression.

Parameters

data	The data that will be used to determine the equation parameters.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 985 of file Trendlines.cs.

7.147.3 Member Function Documentation

7.147.3.1 Plot()

```
void VectSharp.Plots.PowerLawTrendLine.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 988 of file Trendlines.cs.

7.147.4 Property Documentation

7.147.4.1 CoordinateSystem

```
IContinuousCoordinateSystem VectSharp.Plots.PowerLawTrendLine.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 922 of file Trendlines.cs.

7.147.4.2 Intercept

```
double VectSharp.Plots.PowerLawTrendLine.Intercept  [get], [set]
```

The intercept of the trendline (b).

Definition at line 887 of file Trendlines.cs.

7.147.4.3 MaxX

```
double VectSharp.Plots.PowerLawTrendLine.MaxX  [get], [set]
```

The maximum X value for which the trendline is plotted.

Definition at line 902 of file Trendlines.cs.

7.147.4.4 MaxY

```
double VectSharp.Plots.PowerLawTrendLine.MaxY  [get], [set]
```

The maximum Y value for which the trendline is plotted.

Definition at line 907 of file Trendlines.cs.

7.147.4.5 MinX

```
double VectSharp.Plots.PowerLawTrendLine.MinX  [get], [set]
```

The minimum X value for which the trendline is plotted.

Definition at line 892 of file Trendlines.cs.

7.147.4.6 MinY

```
double VectSharp.Plots.PowerLawTrendLine.MinY  [get], [set]
```

The minimum Y value for which the trendline is plotted.

Definition at line 897 of file Trendlines.cs.

7.147.4.7 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.PowerLawTrendLine.PresentationAttributes =
new PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180,
180, 180) }  [get], [set]
```

Presentation attributes for the trendline.

Definition at line 912 of file Trendlines.cs.

7.147.4.8 Slope

```
double VectSharp.Plots.PowerLawTrendLine.Slope  [get], [set]
```

The slope of the trendline (a).

Definition at line 882 of file Trendlines.cs.

7.147.4.9 Tag

```
string VectSharp.Plots.PowerLawTrendLine.Tag  [get], [set]
```

A tag to identify the trendline in the plot.

Definition at line 917 of file Trendlines.cs.

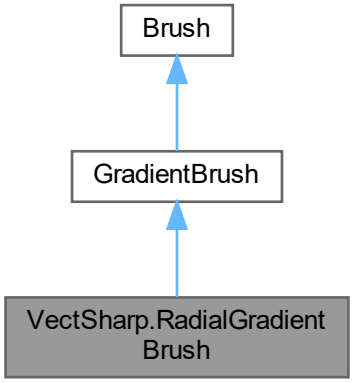
The documentation for this class was generated from the following file:

- VectSharp.Plots/Trendlines.cs

7.148 VectSharp.RadialGradientBrush Class Reference

Represents a brush painting with a radial gradient.

Inheritance diagram for VectSharp.RadialGradientBrush:



Public Member Functions

- RadialGradientBrush (Point focalPoint, Point centre, double radius, params GradientStop[ ] gradientStops)

  Creates a new RadialGradientBrush with the specified focal point, centre, radius and gradient stops.
- RadialGradientBrush (Point focalPoint, Point centre, double radius, IEnumerable< GradientStop > gradient↩ Stops)

  Creates a new RadialGradientBrush with the specified focal point, centre, radius and gradient stops.
- override Brush MultiplyOpacity (double opacity)

  Returns a brush corresponding the current instance, with the specified opacity multiplication applied.

Parameters

opacity	*The value that will be used to multiply the opacity of the brush.*

Returns

A brush corresponding the current instance, with the specified opacity multiplication applied.

Properties

- Point FocalPoint [get]

  The focal point of the gradient (i.e. the point within the circle where the gradient starts).
- Point Centre [get]

  Represents the centre of the gradient.
- double Radius [get]

  The radius of the gradient.

Additional Inherited Members

7.148.1 Detailed Description

Represents a brush painting with a radial gradient.

Definition at line 416 of file Brush.cs.

7.148.2 Constructor & Destructor Documentation

7.148.2.1 RadialGradientBrush() [1/2]

```
VectSharp.RadialGradientBrush.RadialGradientBrush (
            Point focalPoint,
            Point centre,
            double radius,
            params GradientStop[] gradientStops )
```

Creates a new RadialGradientBrush with the specified focal point, centre, radius and gradient stops.

Parameters

focalPoint	The focal point of the gradient. Note that this is relative to the current coordinate system when the gradient is used.
centre	The centre of the gradient. Note that this is relative to the current coordinate system when the gradient is used.
radius	The radius of the gradient. Note that this is relative to the current coordinate system when the gradient is used.
gradientStops	The colour stops in the gradient.

Definition at line 440 of file Brush.cs.

7.148.2.2 RadialGradientBrush() [2/2]

```
VectSharp.RadialGradientBrush.RadialGradientBrush (
            Point focalPoint,
            Point centre,
            double radius,
            IEnumerable< GradientStop > gradientStops )
```

Creates a new RadialGradientBrush with the specified focal point, centre, radius and gradient stops.

Parameters

focalPoint	The focal point of the gradient. Note that this is relative to the current coordinate system when the gradient is used.
centre	The centre of the gradient. Note that this is relative to the current coordinate system when the gradient is used.
radius	The radius of the gradient. Note that this is relative to the current coordinate system when the gradient is used.
gradientStops	The colour stops in the gradient.

Definition at line 479 of file Brush.cs.

7.148.3 Member Function Documentation

7.148.3.1 MultiplyOpacity()

```
override Brush VectSharp.RadialGradientBrush.MultiplyOpacity (
            double opacity )  [virtual]
```

Returns a brush corresponding the current instance, with the specified *opacity* multiplication applied.

Parameters

opacity	The value that will be used to multiply the opacity of the brush.

Returns

A brush corresponding the current instance, with the specified *opacity* multiplication applied.

Implements VectSharp.Brush.

Definition at line 512 of file Brush.cs.

7.148.4 Property Documentation

7.148.4.1 Centre

```
Point VectSharp.RadialGradientBrush.Centre  [get]
```

Represents the centre of the gradient.

Definition at line 426 of file Brush.cs.

7.148.4.2 FocalPoint

```
Point VectSharp.RadialGradientBrush.FocalPoint  [get]
```

The focal point of the gradient (i.e. the point within the circle where the gradient starts).

Definition at line 421 of file Brush.cs.

7.148.4.3 Radius

```
double VectSharp.RadialGradientBrush.Radius  [get]
```

The radius of the gradient.

Definition at line 431 of file Brush.cs.

The documentation for this class was generated from the following file:

- VectSharp/Brush.cs

7.149 VectSharp.TrueTypeFile.CoverageTable.RangeRecord Struct Reference

Public Member Functions

- RangeRecord (Stream fs)

Public Attributes

- ushort StartGlyphID
- ushort EndGlyphID
- ushort StartCoverageIndex

7.149.1 Detailed Description

Definition at line 3804 of file TrueType.cs.

7.149.2 Constructor & Destructor Documentation

7.149.2.1 RangeRecord()

```
VectSharp.TrueTypeFile.CoverageTable.RangeRecord.RangeRecord (
            Stream fs )
```

Definition at line 3810 of file TrueType.cs.

7.149.3 Member Data Documentation

7.149.3.1 EndGlyphID

```
ushort VectSharp.TrueTypeFile.CoverageTable.RangeRecord.EndGlyphID
```

Definition at line 3807 of file TrueType.cs.

7.149.3.2 StartCoverageIndex

```
ushort VectSharp.TrueTypeFile.CoverageTable.RangeRecord.StartCoverageIndex
```

Definition at line 3808 of file TrueType.cs.

7.149.3.3 StartGlyphID

```
ushort VectSharp.TrueTypeFile.CoverageTable.RangeRecord.StartGlyphID
```

Definition at line 3806 of file TrueType.cs.

The documentation for this struct was generated from the following file:

- VectSharp/TrueType.cs

7.150 VectSharp.Raster.Raster Class Reference

Contains methods to render a page to a PNG image.

Static Public Member Functions

- static void SaveAsPNG (this Page page, string fileName, double scale=1)

  Render the page to a PNG file.
- static void SaveAsPNG (this Page page, Stream stream, double scale=1)

  Render the page to a PNG stream.
- static RasterImage Rasterise (this Graphics graphics, Rectangle region, double scale, bool interpolate)

  Rasterise a region of a Graphics object.
- static DisposableIntPtr SaveAsRawBytes (this Page pag, out int width, out int height, out int totalSize, double scale=1)

  Render the page to raw pixel data, in 32bpp RGBA format.
- static void SaveAsAnimatedPNG (this Animation animation, Stream imageStream, double scale=1, double frameRate=60, double durationScaling=1, AnimatedPNG.InterframeCompression interframe↩ Compression=AnimatedPNG.InterframeCompression.First)

  Saves the animation to a stream in animated PNG format.
- static void SaveAsAnimatedPNG (this Animation animation, string fileName, double scale=1, double frameRate=60, double durationScaling=1, AnimatedPNG.InterframeCompression interframe↩ Compression=AnimatedPNG.InterframeCompression.First)

  Saves the animation to an animated PNG file.
- static void SaveAsJPEG (this Page page, string fileName, double scale=1, int quality=90)

  Render the page to a JPEG file.

7.150.1 Detailed Description

Contains methods to render a page to a PNG image.

Definition at line 30 of file Raster.cs.

7.150.2 Member Function Documentation

7.150.2.1 Rasterise()

```
static RasterImage VectSharp.Raster.Raster.Rasterise (
            this Graphics graphics,
            Rectangle region,
            double scale,
            bool interpolate )  [static]
```

Rasterise a region of a Graphics object.

Parameters

graphics	The Graphics object that will be rasterised.
region	The region of the *graphics* that will be rasterised.
scale	The scale at which the image will be rendered.
interpolate	Whether the resulting image should be interpolated or not when it is drawn on another Graphics surface.

Returns

A RasterImage containing the rasterised graphics.

Definition at line 91 of file Raster.cs.

7.150.2.2 SaveAsAnimatedPNG() [1/2]

```
static void VectSharp.Raster.Raster.SaveAsAnimatedPNG (
            this Animation animation,
            Stream imageStream,
            double scale = 1,
            double frameRate = 60,
            double durationScaling = 1,
            AnimatedPNG.InterframeCompression interframeCompression = AnimatedPNG.InterframeCompression.First
)  [static]
```

Saves the animation to a stream in animated PNG format.

Parameters

animation	The animation to export.
imageStream	The stream on which the animated PNG will be written.
scale	The scale at which the animation will be rendered.
frameRate	The target frame rate of the animation, in frames-per-second (fps). This is capped by the animated PNG specification at 90 fps.
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
interframeCompression	The kind of compression that will be used to reduce file size. Note that if the animation has a transparent background, no compression can be performed, and the value of this parameter is ignored.

Definition at line 184 of file Raster.cs.

7.150.2.3 SaveAsAnimatedPNG() [2/2]

```
static void VectSharp.Raster.Raster.SaveAsAnimatedPNG (
            this Animation animation,
            string fileName,
            double scale = 1,
            double frameRate = 60,
            double durationScaling = 1,
            AnimatedPNG.InterframeCompression interframeCompression = AnimatedPNG.InterframeCompression.First
)  [static]
```

Saves the animation to an animated PNG file.

Parameters

animation	The animation to export.
fileName	The output file to create.
scale	The scale at which the animation will be rendered.
frameRate	The target frame rate of the animation, in frames-per-second (fps). This is capped by the animated PNG specification at 90 fps.
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
interframeCompression	The kind of compression that will be used to reduce file size. Note that if the animation has a transparent background, no compression can be performed, and the value of this parameter is ignored.

Definition at line 267 of file Raster.cs.

7.150.2.4 SaveAsJPEG()

```
static void VectSharp.Raster.Raster.SaveAsJPEG (
            this Page page,
            string fileName,
            double scale = 1,
            int quality = 90 )  [static]
```

Render the page to a JPEG file.

Parameters

page	The Page to render.
fileName	The full path to the file to save. If it exists, it will be overwritten.
scale	The scale to be used when rasterising the page. This will determine the width and height of the image file.
quality	The quality of the JPEG output (ranging from 0 to 100).

Definition at line 283 of file Raster.cs.

7.150.2.5 SaveAsPNG() [1/2]

```
static void VectSharp.Raster.Raster.SaveAsPNG (
            this Page page,
            Stream stream,
            double scale = 1 )  [static]
```

Render the page to a PNG stream.

Parameters

page	The Page to render.
stream	The stream to which the PNG data will be written.
scale	The scale to be used when rasterising the page. This will determine the width and height of the image file.

Definition at line 64 of file Raster.cs.

7.150.2.6 SaveAsPNG() [2/2]

```
static void VectSharp.Raster.Raster.SaveAsPNG (
            this Page page,
            string fileName,
            double scale = 1 )  [static]
```

Render the page to a PNG file.

Parameters

page	The Page to render.
fileName	The full path to the file to save. If it exists, it will be overwritten.
scale	The scale to be used when rasterising the page. This will determine the width and height of the image file.

Definition at line 39 of file Raster.cs.

7.150.2.7 SaveAsRawBytes()

```
static DisposableIntPtr VectSharp.Raster.Raster.SaveAsRawBytes (
            this Page pag,
            out int width,
            out int height,
            out int totalSize,
            double scale = 1 )  [static]
```

Render the page to raw pixel data, in 32bpp RGBA format.

Parameters

pag	The Page to render.
scale	The scale to be used when rasterising the page. This will determine the width and height of the image.
width	The width of the rendered image.
height	The height of the rendered image.
totalSize	The size in bytes of the raw pixel data.

Returns

A DisposableIntPtr containing a pointer to the raw pixel data, stored in unmanaged memory. Dispose this object to release the unmanaged memory.

Definition at line 143 of file Raster.cs.

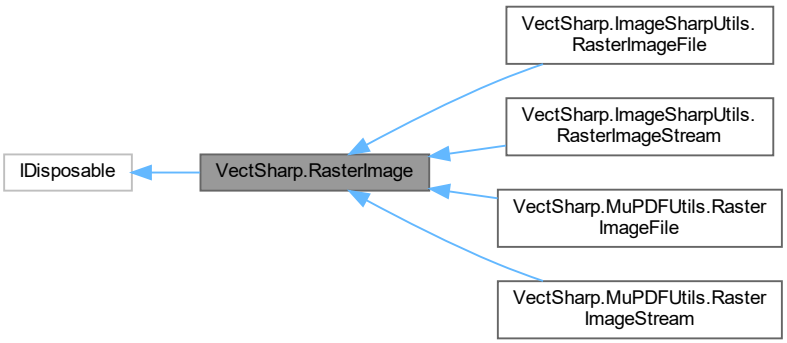
The documentation for this class was generated from the following file:

- VectSharp.Raster/Raster.cs

7.151 VectSharp.RasterImage Class Reference

Represents a raster image, created from raw pixel data. Consider using the derived classes included in the NuGet package "VectSharp.MuPDFUtils" if you need to load a raster image from a file or a Stream.

Inheritance diagram for VectSharp.RasterImage:



Public Member Functions

- RasterImage (IntPtr pixelData, int width, int height, bool hasAlpha, bool interpolate)

  Creates a new RasterImage instance from the specified pixel data in RGB or RGBA format.
- RasterImage (ref DisposableIntPtr pixelData, int width, int height, bool hasAlpha, bool interpolate)

  Creates a new RasterImage instance from the specified pixel data in RGB or RGBA format.
- RasterImage (byte[ ] data, int width, int height, PixelFormats pixelFormat, bool interpolate)

  Creates a new RasterImage instance copying the specified pixel data.
- void ClearPNGCache ()

  Disposes the PNGStream. Also useful if is is necessary to regenerate it, e.g. because the underlying image pixel data has changed.
- void Dispose ()

Properties

- IntPtr ImageDataAddress `[get]`

  The memory address of the image pixel data.
- IDisposable DataHolder `[get]`

  An IDisposable that will be disposed when the image is disposed.
- string Id `[get]`

  A univocal identifier for this image.
- bool HasAlpha `[get]`

  Determines whether the image has an alpha channel.
- int Width `[get]`

  The width in pixels of the image.
- int Height `[get]`

  The height in pixels of the image.
- bool Interpolate `[get]`

  Determines whether the image should be interpolated when it is resized.
- MemoryStream PNGStream `[get]`

  Contains a representation of the image in PNG format. Generated at the first access and cached until the image is disposed.

7.151.1 Detailed Description

Represents a raster image, created from raw pixel data. Consider using the derived classes included in the NuGet package "VectSharp.MuPDFUtils" if you need to load a raster image from a file or a Stream.

Definition at line 100 of file RasterImage.cs.

7.151.2 Constructor & Destructor Documentation

7.151.2.1 RasterImage() `[1/3]`

```
VectSharp.RasterImage.RasterImage (
            IntPtr pixelData,
            int width,
            int height,
            bool hasAlpha,
            bool interpolate )
```

Creates a new RasterImage instance from the specified pixel data in RGB or RGBA format.

Parameters

pixelData	The address of the image pixel data in RGB or RGBA format.
width	The width in pixels of the image.
height	The height in pixels of the image.
hasAlpha	true if the image is in RGBA format, false if it is in RGB format.
interpolate	Whether the image should be interpolated when it is resized.

Definition at line 172 of file RasterImage.cs.

7.151.2.2 RasterImage() [2/3]

```
VectSharp.RasterImage.RasterImage (
            ref DisposableIntPtr pixelData,
            int width,
            int height,
            bool hasAlpha,
            bool interpolate )
```

Creates a new RasterImage instance from the specified pixel data in RGB or RGBA format.

Parameters

pixelData	The address of the image pixel data in RGB or RGBA format wrapped in a DisposableIntPtr. The RasterImage will take ownership of this memory.
width	The width in pixels of the image.
height	The height in pixels of the image.
hasAlpha	true if the image is in RGBA format, false if it is in RGB format.
interpolate	Whether the image should be interpolated when it is resized.

Definition at line 190 of file RasterImage.cs.

7.151.2.3 RasterImage() [3/3]

```
VectSharp.RasterImage.RasterImage (
            byte[] data,
            int width,
            int height,
            PixelFormats pixelFormat,
            bool interpolate )
```

Creates a new RasterImage instance copying the specified pixel data.

Parameters

data	The image pixel data that will be copied.
width	The width in pixels of the image.
height	The height in pixels of the image.
pixelFormat	The format of the pixel data.
interpolate	Whether the image should be interpolated when it is resized.

Definition at line 209 of file RasterImage.cs.

7.151.3 Member Function Documentation

7.151.3.1 ClearPNGCache()

```
void VectSharp.RasterImage.ClearPNGCache ( )
```

Disposes the PNGStream. Also useful if is is necessary to regenerate it, e.g. because the underlying image pixel data has changed.

Definition at line 263 of file RasterImage.cs.

7.151.3.2 Dispose()

```
void VectSharp.RasterImage.Dispose ( )
```

Definition at line 297 of file RasterImage.cs.

7.151.4 Property Documentation

7.151.4.1 DataHolder

```
IDisposable VectSharp.RasterImage.DataHolder  [get]
```

An IDisposable that will be disposed when the image is disposed.

Definition at line 110 of file RasterImage.cs.

7.151.4.2 HasAlpha

```
bool VectSharp.RasterImage.HasAlpha  [get]
```

Determines whether the image has an alpha channel.

Definition at line 120 of file RasterImage.cs.

7.151.4.3 Height

```
int VectSharp.RasterImage.Height  [get]
```

The height in pixels of the image.

Definition at line 130 of file RasterImage.cs.

7.151.4.4 Id

```
string VectSharp.RasterImage.Id  [get]
```

A univocal identifier for this image.

Definition at line 115 of file RasterImage.cs.

7.151.4.5 ImageDataAddress

```
IntPtr VectSharp.RasterImage.ImageDataAddress  [get]
```

The memory address of the image pixel data.

Definition at line 105 of file RasterImage.cs.

7.151.4.6 Interpolate

```
bool VectSharp.RasterImage.Interpolate  [get]
```

Determines whether the image should be interpolated when it is resized.

Definition at line 135 of file RasterImage.cs.

7.151.4.7 PNGStream

```
MemoryStream VectSharp.RasterImage.PNGStream  [get]
```

Contains a representation of the image in PNG format. Generated at the first access and cached until the image is disposed.

Definition at line 142 of file RasterImage.cs.

7.151.4.8 Width

```
int VectSharp.RasterImage.Width  [get]
```

The width in pixels of the image.

Definition at line 125 of file RasterImage.cs.

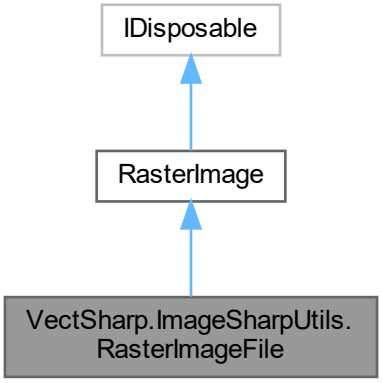
The documentation for this class was generated from the following file:

- VectSharp/RasterImage.cs

7.152 VectSharp.ImageSharpUtils.RasterImageFile Class Reference

A RasterImage created from a file.

Inheritance diagram for VectSharp.ImageSharpUtils.RasterImageFile:



Public Member Functions

- RasterImageFile (string fileName, bool alpha=true, bool interpolate=true)

  Creates a new RasterImage from the specified file.

Additional Inherited Members

7.152.1 Detailed Description

A RasterImage created from a file.

Definition at line 30 of file RasterImages.cs.

7.152.2 Constructor & Destructor Documentation

7.152.2.1 RasterImageFile()

```
VectSharp.ImageSharpUtils.RasterImageFile.RasterImageFile (
            string fileName,
            bool alpha = true,
            bool interpolate = true )
```

Creates a new RasterImage from the specified file.

Parameters

fileName	The path to the file containing the image.
alpha	A boolean value indicating whether transparency (alpha) data from the image should be preserved or not.
interpolate	A boolean value indicating whether the image should be interpolated when it is resized or not.

Definition at line 38 of file RasterImages.cs.

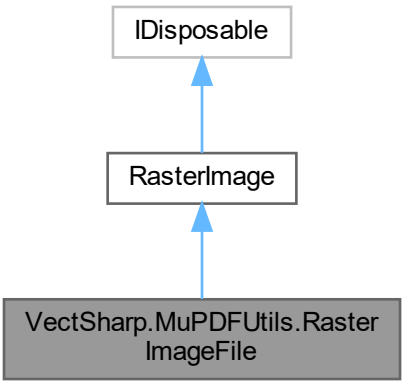
The documentation for this class was generated from the following file:

- VectSharp.ImageSharpUtils/RasterImages.cs

7.153 VectSharp.MuPDFUtils.RasterImageFile Class Reference

A RasterImage created from a file.

Inheritance diagram for VectSharp.MuPDFUtils.RasterImageFile:



Public Member Functions

- RasterImageFile (string fileName, int pageNumber=0, double scale=1, bool alpha=true, bool interpolate=true)

  Creates a new RasterImage from the specified file.

Additional Inherited Members

7.153.1 Detailed Description

A RasterImage created from a file.

Definition at line 28 of file RasterImages.cs.

7.153.2 Constructor & Destructor Documentation

7.153.2.1 RasterImageFile()

```
VectSharp.MuPDFUtils.RasterImageFile.RasterImageFile (
            string fileName,
            int pageNumber = 0,
            double scale = 1,
            bool alpha = true,
            bool interpolate = true )
```

Creates a new RasterImage from the specified file.

Parameters

fileName	The path to the file containing the image.
pageNumber	The number of the page in the file from which the image should be created, starting at 0. Only useful for multi-page formats, such as PDF.
scale	The scale factor at which to render the image.
alpha	A boolean value indicating whether transparency (alpha) data from the image should be preserved or not.
interpolate	A boolean value indicating whether the image should be interpolated when it is resized or not.

Definition at line 38 of file RasterImages.cs.

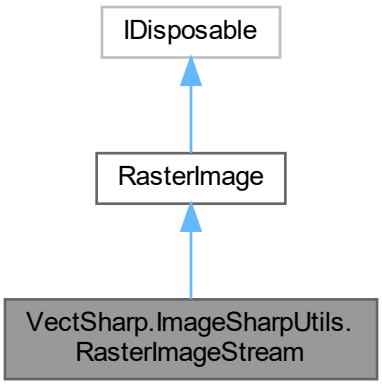
The documentation for this class was generated from the following file:

- VectSharp.MuPDFUtils/RasterImages.cs

7.154 VectSharp.ImageSharpUtils.RasterImageStream Class Reference

A RasterImage created from a stream.

Inheritance diagram for VectSharp.ImageSharpUtils.RasterImageStream:



Public Member Functions

- RasterImageStream (Stream imageStream, bool alpha=true, bool interpolate=true)

  Creates a new RasterImage from the specified stream.
- RasterImageStream (IntPtr imageAddress, int imageLength, bool alpha=true, bool interpolate=true)

  Creates a new RasterImage from the specified stream.

Additional Inherited Members

7.154.1 Detailed Description

A RasterImage created from a stream.

Definition at line 106 of file RasterImages.cs.

7.154.2 Constructor & Destructor Documentation

7.154.2.1 RasterImageStream() [1/2]

```
VectSharp.ImageSharpUtils.RasterImageStream.RasterImageStream (
            Stream imageStream,
            bool alpha = true,
            bool interpolate = true )
```

Creates a new RasterImage from the specified stream.

Parameters

imageStream	The stream containing the image data.
alpha	A boolean value indicating whether transparency (alpha) data from the image should be preserved or not.
interpolate	A boolean value indicating whether the image should be interpolated when it is resized or not.

Definition at line 114 of file RasterImages.cs.

7.154.2.2 RasterImageStream() [2/2]

```
VectSharp.ImageSharpUtils.RasterImageStream.RasterImageStream (
            IntPtr imageAddress,
            int imageLength,
            bool alpha = true,
            bool interpolate = true )
```

Creates a new RasterImage from the specified stream.

Parameters

imageAddress	A pointer to the address where the image data is contained.
imageLength	The length in bytes of the image data.
alpha	A boolean value indicating whether transparency (alpha) data from the image should be preserved or not.
interpolate	A boolean value indicating whether the image should be interpolated when it is resized or not.

Definition at line 185 of file RasterImages.cs.

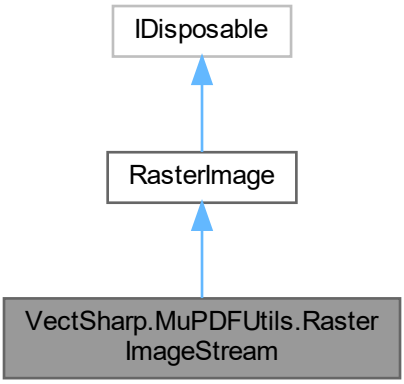
The documentation for this class was generated from the following file:

- VectSharp.ImageSharpUtils/RasterImages.cs

7.155 VectSharp.MuPDFUtils.RasterImageStream Class Reference

A RasterImage created from a stream.

Inheritance diagram for VectSharp.MuPDFUtils.RasterImageStream:



Public Member Functions

- RasterImageStream (Stream imageStream, InputFileTypes fileType, int pageNumber=0, double scale=1, bool alpha=true, bool interpolate=true)

  Creates a new RasterImage from the specified stream.
- RasterImageStream (IntPtr imageAddress, long imageLength, InputFileTypes fileType, int pageNumber=0, double scale=1, bool alpha=true, bool interpolate=true)

  Creates a new RasterImage from the specified stream.

Additional Inherited Members

7.155.1 Detailed Description

A RasterImage created from a stream.

Definition at line 69 of file RasterImages.cs.

7.155.2 Constructor & Destructor Documentation

7.155.2.1 RasterImageStream() [1/2]

```
VectSharp.MuPDFUtils.RasterImageStream.RasterImageStream (
            Stream imageStream,
            InputFileTypes fileType,
            int pageNumber = 0,
            double scale = 1,
            bool alpha = true,
            bool interpolate = true )
```

Creates a new RasterImage from the specified stream.

Parameters

imageStream	The stream containing the image data.
fileType	The type of the image contained in the stream.
pageNumber	The number of the page in the file from which the image should be created, starting at 0. Only useful for multi-page formats, such as PDF.
scale	The scale factor at which to render the image.
alpha	A boolean value indicating whether transparency (alpha) data from the image should be preserved or not.
interpolate	A boolean value indicating whether the image should be interpolated when it is resized or not.

Definition at line 80 of file RasterImages.cs.

7.155.2.2 RasterImageStream() [2/2]

```
VectSharp.MuPDFUtils.RasterImageStream.RasterImageStream (
            IntPtr imageAddress,
            long imageLength,
            InputFileTypes fileType,
            int pageNumber = 0,
            double scale = 1,
            bool alpha = true,
            bool interpolate = true )
```

Creates a new RasterImage from the specified stream.

Parameters

imageAddress	A pointer to the address where the image data is contained.
imageLength	The length in bytes of the image data.
fileType	The type of the image contained in the stream.
pageNumber	The number of the page in the file from which the image should be created, starting at 0. Only useful for multi-page formats, such as PDF.
scale	The scale factor at which to render the image.
alpha	A boolean value indicating whether transparency (alpha) data from the image should be preserved or not.
interpolate	A boolean value indicating whether the image should be interpolated when it is resized or not.

Definition at line 148 of file RasterImages.cs.

The documentation for this class was generated from the following file:

- VectSharp.MuPDFUtils/RasterImages.cs

7.156 VectSharp.Rectangle Struct Reference

Represents a rectangle.

Public Member Functions

- bool IsNaN ()

  Checks whether the rectangle is equivalent to Rectangle.NaN.
- Rectangle (Point location, Size size)

  Create a new Rectangle given its top-left corner and its size.
- Rectangle (double x, double y, double width, double height)

  Create a new Rectangle given its top-left corner and its size.
- Rectangle (Point topLeft, Point bottomRight)

  Create a new Rectangle given its top-left corner and its bottom-right corner.

Static Public Member Functions

- static Rectangle Union (Rectangle rectangle1, Rectangle rectangle2)

  Computes the rectangular bounds of the union of two Rectangles.
- static Rectangle Intersection (Rectangle rectangle1, Rectangle rectangle2)

  Computes the intersection of two Rectangles.
- static Rectangle Union (IEnumerable< Rectangle > rectangles)

  Computes the rectangular bounds of the union of multiple Rectangles.
- static Rectangle Union (params Rectangle[ ] rectangles)

  Computes the rectangular bounds of the union of multiple Rectangles.

Public Attributes

- Point Location

  The top-left corner of the rectangle.
- Size Size

  The size of the rectangle.

Static Public Attributes

- static readonly Rectangle NaN = new Rectangle(double.NaN, double.NaN, double.NaN, double.NaN)

  A rectangle whose dimensions are all double.NaN.

Properties

- Point Centre `[get]`

  The centre of the rectangle.

7.156.1 Detailed Description

Represents a rectangle.

Definition at line 301 of file Point.cs.

7.156.2 Constructor & Destructor Documentation

7.156.2.1 Rectangle() [1/3]

```
VectSharp.Rectangle.Rectangle (
            Point location,
            Size size )
```

Create a new Rectangle given its top-left corner and its size.

Parameters

location	The top-left corner of the rectangle.
size	The size of the rectangle.

Definition at line 337 of file Point.cs.

7.156.2.2 Rectangle() [2/3]

```
VectSharp.Rectangle.Rectangle (
            double x,
            double y,
            double width,
            double height )
```

Create a new Rectangle given its top-left corner and its size.

Parameters

x	The horizontal coordinate of the top-left corner of the rectangle.
y	The vertical coordinate of the top-left corner of the rectangle.
width	The width of the rectangle.
height	The height of the rectangle.

Definition at line 350 of file Point.cs.

7.156.2.3 Rectangle() [3/3]

```
VectSharp.Rectangle.Rectangle (
            Point topLeft,
            Point bottomRight )
```

Create a new Rectangle given its top-left corner and its bottom-right corner.

Parameters

topLeft	The top-left corner of the rectangle.
bottomRight	The bottom-right corner of the rectangle.

Definition at line 361 of file Point.cs.

7.156.3 Member Function Documentation

7.156.3.1 Intersection()

```
static Rectangle VectSharp.Rectangle.Intersection (
            Rectangle rectangle1,
            Rectangle rectangle2 )  [static]
```

Computes the intersection of two Rectangles.

Parameters

rectangle1	The first Rectangle.
rectangle2	The second Rectangle.

Returns

The rectangle corresponding to the intersection of *rectangle1* and *rectangle2* , or Rectangle.NaN if the intersection is empty.

Definition at line 406 of file Point.cs.

7.156.3.2 IsNaN()

```
bool VectSharp.Rectangle.IsNaN ( )
```

Checks whether the rectangle is equivalent to Rectangle.NaN.

Returns

`true` if all the dimensions of the rectangle are double.NaN, `false` otherwise.

Definition at line 312 of file Point.cs.

7.156.3.3 Union() [1/3]

```
static Rectangle VectSharp.Rectangle.Union (
            IEnumerable< Rectangle > rectangles )  [static]
```

Computes the rectangular bounds of the union of multiple Rectangles.

Parameters

rectangles	The Rectangles whose union will be computed.

Returns

The smallest Rectangle containing all the *rectangles* .

Definition at line 430 of file Point.cs.

7.156.3.4 Union() [2/3]

```
static Rectangle VectSharp.Rectangle.Union (
            params Rectangle[] rectangles )  [static]
```

Computes the rectangular bounds of the union of multiple Rectangles.

Parameters

rectangles	The Rectangles whose union will be computed.

Returns

The smallest Rectangle containing all the *rectangles* .

Definition at line 463 of file Point.cs.

7.156.3.5 Union() [3/3]

```
static Rectangle VectSharp.Rectangle.Union (
            Rectangle rectangle1,
            Rectangle rectangle2 )  [static]
```

Computes the rectangular bounds of the union of two Rectangles.

Parameters

rectangle1	The first Rectangle.
rectangle2	The second Rectangle.

Returns

The smallest Rectangle containing both *rectangle1* and *rectangle2* .

Definition at line 373 of file Point.cs.

7.156.4 Member Data Documentation

7.156.4.1 Location

```
Point VectSharp.Rectangle.Location
```

The top-left corner of the rectangle.

Definition at line 320 of file Point.cs.

7.156.4.2 NaN

```
readonly Rectangle VectSharp.Rectangle.NaN = new Rectangle(double.NaN, double.NaN, double.NaN,
double.NaN)  [static]
```

A rectangle whose dimensions are all double.NaN.

Definition at line 306 of file Point.cs.

7.156.4.3 Size

```
Size VectSharp.Rectangle.Size
```

The size of the rectangle.

Definition at line 325 of file Point.cs.

7.156.5 Property Documentation

7.156.5.1 Centre

```
Point VectSharp.Rectangle.Centre  [get]
```

The centre of the rectangle.

Definition at line 330 of file Point.cs.

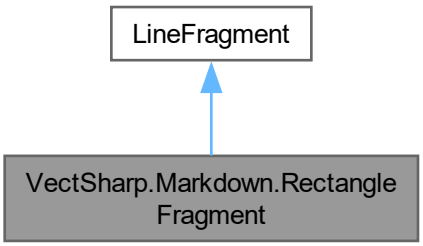
The documentation for this struct was generated from the following file:

- VectSharp/Point.cs

7.157 VectSharp.Markdown.RectangleFragment Class Reference

Represents a rectangle.

Inheritance diagram for VectSharp.Markdown.RectangleFragment:



Public Member Functions

- RectangleFragment (Point topLeft, Size size, Colour colour, string tag)

  Create a new RectangleFragment.
- override void Translate (double deltaX, double deltaY)

  Applies the specified translation to the line fragment.

Parameters

deltaX	*The translation in the horizontal direction.*
deltaY	*The translation in the vertical direction.*

- override double GetAscent (double lineAscent)

  Get the ascent of the LineFragment.

Parameters

lineAscent	*The ascent of the Line on which the fragment should be drawn.*

Returns

The ascent of the LineFragment.

- override double GetMaxY ()

  Get the vertical coordinate of the bottom of the LineFragment.

  Returns

  The vertical coordinates of the bottom of the LineFragment.
- override double GetMaxX ()

  Get the horizontal coordinate of the right side of the LineFragment.

  Returns

  The horizontal coordinate of the right side of the LineFragment.
- override? Point Render (Graphics graphics, double deltaY)

  Render the LineFragment on the specified graphics surface.

Parameters

graphics	*The Graphics surface on which the LineFragment will be drawn.*
deltaY	*Additional translation on the vertical axis.*

Returns

If a text element has been drawn, this method should return the bottom-right corner of the drawn text element, otherwise `null`*. This is used to update the MarkdownContext.BottomRight property.*

Properties

- Point TopLeft `[get]`

  The top-left corner of the rectangle, relative to the cursor at the start of the line.
- Size Size `[get]`

  The size of the rectangle.
- Colour Colour `[get]`

  The colour used to draw the rectangle.

7.157.1 Detailed Description

Represents a rectangle.

Definition at line 222 of file Line.cs.

7.157.2 Constructor & Destructor Documentation

7.157.2.1 RectangleFragment()

```
VectSharp.Markdown.RectangleFragment.RectangleFragment (
            Point topLeft,
            Size size,
            Colour colour,
            string tag )
```

Create a new RectangleFragment.

Parameters

topLeft	The top-left corner of the rectangle.
size	The size of the rectangle.
colour	The colour used to draw the rectangle.
tag	Graphics action tag for the rectangle.

Definition at line 246 of file Line.cs.

7.157.3 Member Function Documentation

7.157.3.1 GetAscent()

```
override double VectSharp.Markdown.RectangleFragment.GetAscent (
            double lineAscent )  [virtual]
```

Get the ascent of the LineFragment.

Parameters

lineAscent	The ascent of the Line on which the fragment should be drawn.

Returns

The ascent of the LineFragment.

Implements VectSharp.Markdown.LineFragment.

Definition at line 261 of file Line.cs.

7.157.3.2 GetMaxX()

```
override double VectSharp.Markdown.RectangleFragment.GetMaxX ( )  [virtual]
```

Get the horizontal coordinate of the right side of the LineFragment.

Returns

The horizontal coordinate of the right side of the LineFragment.

Implements VectSharp.Markdown.LineFragment.

Definition at line 273 of file Line.cs.

7.157.3.3 GetMaxY()

```
override double VectSharp.Markdown.RectangleFragment.GetMaxY ( )  [virtual]
```

Get the vertical coordinate of the bottom of the LineFragment.

Returns

The vertical coordinates of the bottom of the LineFragment.

Implements VectSharp.Markdown.LineFragment.

Definition at line 267 of file Line.cs.

7.157.3.4 Render()

```
override?  Point VectSharp.Markdown.RectangleFragment.Render (
             Graphics graphics,
             double deltaY )  [virtual]
```

Render the LineFragment on the specified *graphics* surface.

Parameters

graphics	The Graphics surface on which the LineFragment will be drawn.
deltaY	Additional translation on the vertical axis.

Returns

If a text element has been drawn, this method should return the bottom-right corner of the drawn text element, otherwise `null`. This is used to update the MarkdownContext.BottomRight property.

Implements VectSharp.Markdown.LineFragment.

Definition at line 279 of file Line.cs.

7.157.3.5 Translate()

```
override void VectSharp.Markdown.RectangleFragment.Translate (
             double deltaX,
             double deltaY )  [virtual]
```

Applies the specified translation to the line fragment.

Parameters

deltaX	The translation in the horizontal direction.
deltaY	The translation in the vertical direction.

Implements VectSharp.Markdown.LineFragment.

Definition at line 255 of file Line.cs.

7.157.4 Property Documentation

7.157.4.1 Colour

```
Colour VectSharp.Markdown.RectangleFragment.Colour  [get]
```

The colour used to draw the rectangle.

Definition at line 237 of file Line.cs.

7.157.4.2 Size

```
Size VectSharp.Markdown.RectangleFragment.Size [get]
```

The size of the rectangle.

Definition at line 232 of file Line.cs.

7.157.4.3 TopLeft

```
Point VectSharp.Markdown.RectangleFragment.TopLeft [get]
```

The top-left corner of the rectangle, relative to the cursor at the start of the line.

Definition at line 227 of file Line.cs.

The documentation for this class was generated from the following file:

- VectSharp.Markdown/Line.cs

7.158 VectSharp.Canvas.RenderAction Class Reference

Represents a light-weight rendering action.

Public Types

- enum ActionTypes

  Types of rendering actions.

Public Member Functions

- void BringToFront ()

  Brings the render action to the front of the rendering queue. This method can only be invoked after the output has been fully initialised.
- void SendToBack ()

  Brings the render action to the back of the rendering queue. This method can only be invoked after the output has been fully initialised.

Static Public Member Functions

- static RenderAction PathAction (Geometry geometry, Pen stroke, IBrush fill, Avalonia.Matrix transform, Geometry clippingPath, string tag=null)

  Creates a new RenderAction representing a path.
- static RenderAction TextAction (Avalonia.Media.FormattedText text, Avalonia.Media.TextFormatting.Text↩ Layout layout, IBrush fill, Avalonia.Matrix transform, Geometry clippingPath, string tag=null)

  Creates a new RenderAction representing text.
- static RenderAction ImageAction (string imageId, Avalonia.Rect sourceRect, Avalonia.Rect destinationRect, Avalonia.Matrix transform, Geometry clippingPath, string tag=null)

  Creates a new RenderAction representing an image.

Properties

- ActionTypes ActionType `[get]`

  Type of the rendering action.
- Geometry Geometry `[get, set]`

  Geometry that needs to be rendered (null if the action type is ActionTypes.Text). If you change this, you need to invalidate the Parent's visual.
- Avalonia.Media.FormattedText Text `[get, set]`

  Text that needs to be rendered (null if the action type is ActionTypes.Path). If you change this, you need to invalidate the Parent's visual and you should also change the Layout property.
- Avalonia.Media.TextFormatting.TextLayout Layout `[get, set]`

  Text layout, used for hit testing (null if the action type is ActionTypes.Path). If you change this, you need to invalidate the Parent's visual and you should also change the Text property.
- Pen Stroke `[get, set]`

  Rendering stroke (null if the action type is ActionTypes.Text or if the rendered action only has a Fill). If you change this, you need to invalidate the Parent's visual.
- IBrush Fill `[get, set]`

  Rendering fill (null if the rendered action only has a Stroke). If you change this, you need to invalidate the Parent's visual.
- string ImageId `[get, set]`

  Univocal identifier of the image that needs to be drawn.
- Avalonia.? Rect ImageSource `[get, set]`

  The source rectangle of the image.
- Avalonia.? Rect ImageDestination `[get, set]`

  The destination rectangle of the image.
- Geometry ClippingPath `[get, set]`

  The current clipping path.
- Avalonia.Matrix InverseTransform = Avalonia.Matrix.Identity `[get]`

  Inverse transformation matrix.
- Avalonia.Matrix Transform `[get, set]`

  Rendering transformation matrix. If you change this, you need to invalidate the Parent's visual.
- string Tag `[get, set]`

  A tag to access the RenderAction.
- Control Parent `[get]`

  The container of this RenderAction.

Events

- EventHandler< Avalonia.Input.PointerEventArgs > PointerEntered

  Raised when the pointer enters the area covered by the RenderAction.
- EventHandler< Avalonia.Input.PointerEventArgs > PointerExited

  Raised when the pointer leaves the area covered by the RenderAction.
- EventHandler< Avalonia.Input.PointerPressedEventArgs > PointerPressed

  Raised when the pointer is pressed while over the area covered by the RenderAction.
- EventHandler< Avalonia.Input.PointerReleasedEventArgs > PointerReleased

  Raised when the pointer is released after a PointerPressed event.

7.158.1 Detailed Description

Represents a light-weight rendering action.

Definition at line 1354 of file AvaloniaContext.cs.

7.158.2 Member Enumeration Documentation

7.158.2.1 ActionTypes

```
enum VectSharp.Canvas.RenderAction.ActionTypes
```

Types of rendering actions.

Definition at line 1359 of file AvaloniaContext.cs.

7.158.3 Member Function Documentation

7.158.3.1 BringToFront()

```
void VectSharp.Canvas.RenderAction.BringToFront ( )
```

Brings the render action to the front of the rendering queue. This method can only be invoked after the output has been fully initialised.

Definition at line 1607 of file AvaloniaContext.cs.

7.158.3.2 ImageAction()

```
static RenderAction VectSharp.Canvas.RenderAction.ImageAction (
            string imageId,
            Avalonia.Rect sourceRect,
            Avalonia.Rect destinationRect,
            Avalonia.Matrix transform,
            Geometry clippingPath,
            string tag = null )  [static]
```

Creates a new RenderAction representing an image.

Parameters

imageId	The univocal identifier of the image to draw.
sourceRect	The source rectangle of the image.
destinationRect	The destination rectangle of the image.
transform	The transform that will be applied to the image.
clippingPath	The clipping path.
tag	A tag to access the RenderAction. If this is null this RenderAction is not visible in the hit test.

Returns

A new RenderAction representing an image.

Definition at line 1590 of file AvaloniaContext.cs.

7.158.3.3 PathAction()

```
static RenderAction VectSharp.Canvas.RenderAction.PathAction (
            Geometry geometry,
            Pen stroke,
            IBrush fill,
            Avalonia.Matrix transform,
            Geometry clippingPath,
            string tag = null )  [static]
```

Creates a new RenderAction representing a path.

Parameters

geometry	The geometry to be rendered.
stroke	The stroke of the path (can be null).
fill	The fill of the path (can be null).
transform	The transform that will be applied to the path.
clippingPath	The clipping path.
tag	A tag to access the RenderAction. If this is null this RenderAction is not visible in the hit test.

Returns

A new RenderAction representing a path.

Definition at line 1541 of file AvaloniaContext.cs.

7.158.3.4 SendToBack()

```
void VectSharp.Canvas.RenderAction.SendToBack ( )
```

Brings the render action to the back of the rendering queue. This method can only be invoked after the output has been fully initialised.

Definition at line 1615 of file AvaloniaContext.cs.

7.158.3.5 TextAction()

```
static RenderAction VectSharp.Canvas.RenderAction.TextAction (
            Avalonia.Media.FormattedText text,
            Avalonia.Media.TextFormatting.TextLayout layout,
            IBrush fill,
            Avalonia.Matrix transform,
            Geometry clippingPath,
            string tag = null )  [static]
```

Creates a new RenderAction representing text.

Parameters

text	The text to be rendered.
layout	The text layout used for hit testing.
fill	The fill of the text (can be null).
transform	The transform that will be applied to the text.
clippingPath	The clipping path.
tag	A tag to access the RenderAction. If this is null this RenderAction is not visible in the hit test.

Returns

A new RenderAction representing text.

Definition at line 1565 of file AvaloniaContext.cs.

7.158.4 Property Documentation

7.158.4.1 ActionType

```
ActionTypes VectSharp.Canvas.RenderAction.ActionType  [get]
```

Type of the rendering action.

Definition at line 1380 of file AvaloniaContext.cs.

7.158.4.2 ClippingPath

```
Geometry VectSharp.Canvas.RenderAction.ClippingPath  [get], [set]
```

The current clipping path.

Definition at line 1445 of file AvaloniaContext.cs.

7.158.4.3 Fill

```
IBrush VectSharp.Canvas.RenderAction.Fill  [get], [set]
```

Rendering fill (null if the rendered action only has a Stroke). If you change this, you need to invalidate the Parent's visual.

Definition at line 1410 of file AvaloniaContext.cs.

7.158.4.4 Geometry

```
Geometry VectSharp.Canvas.RenderAction.Geometry  [get], [set]
```

Geometry that needs to be rendered (null if the action type is ActionTypes.Text). If you change this, you need to invalidate the Parent's visual.

Definition at line 1385 of file AvaloniaContext.cs.

7.158.4.5 ImageDestination

```
Avalonia.?  Rect VectSharp.Canvas.RenderAction.ImageDestination  [get], [set]
```

The destination rectangle of the image.

Definition at line 1440 of file AvaloniaContext.cs.

7.158.4.6 ImageId

```
string VectSharp.Canvas.RenderAction.ImageId  [get], [set]
```

Univocal identifier of the image that needs to be drawn.

Definition at line 1430 of file AvaloniaContext.cs.

7.158.4.7 ImageSource

```
Avalonia.?  Rect VectSharp.Canvas.RenderAction.ImageSource  [get], [set]
```

The source rectangle of the image.

Definition at line 1435 of file AvaloniaContext.cs.

7.158.4.8 InverseTransform

```
Avalonia.Matrix VectSharp.Canvas.RenderAction.InverseTransform = Avalonia.Matrix.Identity
[get]
```

Inverse transformation matrix.

Definition at line 1452 of file AvaloniaContext.cs.

7.158.4.9 Layout

```
Avalonia.Media.TextFormatting.TextLayout VectSharp.Canvas.RenderAction.Layout  [get], [set]
```

Text layout, used for hit testing (null if the action type is ActionTypes.Path). If you change this, you need to invalidate the Parent's visual and you should also change the Text property.

Definition at line 1397 of file AvaloniaContext.cs.

7.158.4.10 Parent

```
Control VectSharp.Canvas.RenderAction.Parent  [get]
```

The container of this RenderAction.

Definition at line 1477 of file AvaloniaContext.cs.

7.158.4.11 Stroke

```
Pen VectSharp.Canvas.RenderAction.Stroke  [get], [set]
```

Rendering stroke (null if the action type is ActionTypes.Text or if the rendered action only has a Fill). If you change this, you need to invalidate the Parent's visual.

Definition at line 1402 of file AvaloniaContext.cs.

7.158.4.12 Tag

```
string VectSharp.Canvas.RenderAction.Tag  [get], [set]
```

A tag to access the RenderAction.

Definition at line 1470 of file AvaloniaContext.cs.

7.158.4.13 Text

```
Avalonia.Media.FormattedText VectSharp.Canvas.RenderAction.Text  [get], [set]
```

Text that needs to be rendered (null if the action type is ActionTypes.Path). If you change this, you need to invalidate the Parent's visual and you should also change the Layout property.

Definition at line 1391 of file AvaloniaContext.cs.

7.158.4.14 Transform

```
Avalonia.Matrix VectSharp.Canvas.RenderAction.Transform  [get], [set]
```

Rendering transformation matrix. If you change this, you need to invalidate the Parent's visual.

Definition at line 1457 of file AvaloniaContext.cs.

7.158.5 Event Documentation

7.158.5.1 PointerEntered

```
EventHandler<Avalonia.Input.PointerEventArgs> VectSharp.Canvas.RenderAction.PointerEntered
```

Raised when the pointer enters the area covered by the RenderAction.

Definition at line 1488 of file AvaloniaContext.cs.

7.158.5.2 PointerExited

```
EventHandler<Avalonia.Input.PointerEventArgs> VectSharp.Canvas.RenderAction.PointerExited
```

Raised when the pointer leaves the area covered by the RenderAction.

Definition at line 1493 of file AvaloniaContext.cs.

7.158.5.3 PointerPressed

```
EventHandler<Avalonia.Input.PointerPressedEventArgs> VectSharp.Canvas.RenderAction.Pointer↩
Pressed
```

Raised when the pointer is pressed while over the area covered by the RenderAction.

Definition at line 1498 of file AvaloniaContext.cs.

7.158.5.4 PointerReleased

```
EventHandler<Avalonia.Input.PointerReleasedEventArgs> VectSharp.Canvas.RenderAction.Pointer↩
Released
```

Raised when the pointer is released after a PointerPressed event.

Definition at line 1503 of file AvaloniaContext.cs.

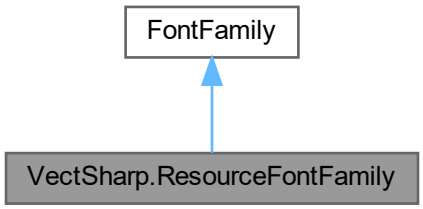
The documentation for this class was generated from the following file:

- VectSharp.Canvas/AvaloniaContext.cs

7.159 VectSharp.ResourceFontFamily Class Reference

Represents a FontFamily created from a resource stream.

Inheritance diagram for VectSharp.ResourceFontFamily:



Public Member Functions

- ResourceFontFamily (System.IO.Stream resourceStream, string resourceName)

  Create a new ResourceFontFamily from the specified resourceStream containing a TTF file, passing the specified resourceName to the

Public Attributes

- string ResourceName

  The name of the embedded resource, which will be parsed using

Additional Inherited Members

7.159.1 Detailed Description

Represents a FontFamily created from a resource stream.

Definition at line 697 of file Font.cs.

7.159.2 Constructor & Destructor Documentation

7.159.2.1 ResourceFontFamily()

```
VectSharp.ResourceFontFamily.ResourceFontFamily (
            System.IO.Stream resourceStream,
            string resourceName )
```

Create a new ResourceFontFamily from the specified *resourceStream* containing a TTF file, passing the specified *resourceName* to the

```
Avalonia.Media.FontFamily.Parse(string, Uri)"</code> method.  </summary>
<param name="resourceStream">A resource stream containing a TTF file.</param>
<param name="resourceName">The name of the embedded resource, which will be
parsed using Avalonia.Media.FontFamily.Parse(string, Uri).
```

Definition at line 709 of file Font.cs.

7.159.3 Member Data Documentation

7.159.3.1 ResourceName

```
string VectSharp.ResourceFontFamily.ResourceName
```

The name of the embedded resource, which will be parsed using

```
Avalonia.Media.FontFamily.Parse(string, Uri).
```

Definition at line 702 of file Font.cs.

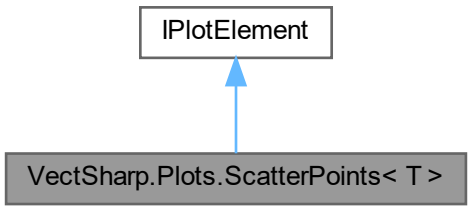
The documentation for this class was generated from the following file:

- VectSharp/Font.cs

7.160 VectSharp.Plots.ScatterPoints< T > Class Template Reference

A plot element that draws a symbol at the location of multiple data points.

Inheritance diagram for VectSharp.Plots.ScatterPoints< T >:



Public Member Functions

- ScatterPoints (IEnumerable< T > data, ICoordinateSystem< T > coordinateSystem)

  Creates a new ScatterPoints< T > instance, using the specified data and coordinateSystem .
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IEnumerable< T > Data `[get, set]`

  The data points at which the symbols will be drawn.
- double Size = 2 `[get, set]`

  The size of the symbols to draw, in plot coordinates.
- IDataPointElement DataPointElement = new PathDataPointElement() `[get, set]`

  The symbol that will be drawn (by default, a circle).
- ICoordinateSystem< T > CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() `[get, set]`

  Presentation attributes determining the appearance (stroke and fill colour, etc.) of the symbols.
- string Tag `[get, set]`

  A tag to identify the symbols in the plot.

7.160.1 Detailed Description

A plot element that draws a symbol at the location of multiple data points.

Template Parameters

T	The kind of data describing the data points (generally, `IReadOnlyList<double>`).

Definition at line 150 of file DataPoints.cs.

7.160.2 Constructor & Destructor Documentation

7.160.2.1 ScatterPoints()

```
VectSharp.Plots.ScatterPoints< T >.ScatterPoints (
            IEnumerable< T > data,
            ICoordinateSystem< T > coordinateSystem )
```

Creates a new ScatterPoints<T> instance, using the specified *data* and *coordinateSystem* .

Parameters

data	The data points at which the symbols will be drawn.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 188 of file DataPoints.cs.

7.160.3 Member Function Documentation

7.160.3.1 Plot()

```
void VectSharp.Plots.ScatterPoints< T >.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 195 of file DataPoints.cs.

7.160.4 Property Documentation

7.160.4.1 CoordinateSystem

```
ICoordinateSystem<T> VectSharp.Plots.ScatterPoints< T >.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 170 of file DataPoints.cs.

7.160.4.2 Data

```
IEnumerable<T> VectSharp.Plots.ScatterPoints< T >.Data  [get], [set]
```

The data points at which the symbols will be drawn.

Definition at line 155 of file DataPoints.cs.

7.160.4.3 DataPointElement

```
IDataPointElement VectSharp.Plots.ScatterPoints< T >.DataPointElement = new PathDataPointElement()
[get], [set]
```

The symbol that will be drawn (by default, a circle).

Definition at line 165 of file DataPoints.cs.

7.160.4.4 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.ScatterPoints< T >.PresentationAttributes =
new PlotElementPresentationAttributes()  [get], [set]
```

Presentation attributes determining the appearance (stroke and fill colour, etc.) of the symbols.

Definition at line 176 of file DataPoints.cs.

7.160.4.5 Size

```
double VectSharp.Plots.ScatterPoints< T >.Size = 2  [get], [set]
```

The size of the symbols to draw, in plot coordinates.

Definition at line 160 of file DataPoints.cs.

7.160.4.6 Tag

```
string VectSharp.Plots.ScatterPoints< T >.Tag  [get], [set]
```

A tag to identify the symbols in the plot.

Definition at line 181 of file DataPoints.cs.

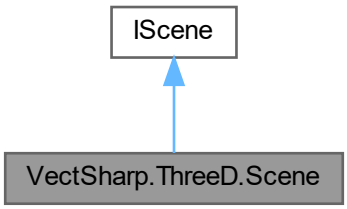
The documentation for this class was generated from the following file:

- VectSharp.Plots/DataPoints.cs

7.161 VectSharp.ThreeD.Scene Class Reference

Represents a 3D scene.

Inheritance diagram for VectSharp.ThreeD.Scene:



Public Member Functions

- Scene ()

  Creates a new Scene.
- void AddElement (Element3D element)

  Adds the specified element to the scene.

Parameters

element	*The Element3D to add.*

- void AddRange (IEnumerable< Element3D > elements)

  Adds the specified elements to the scene.

Parameters

elements	*A collection of Element3Ds to add.*

- void Replace (Func< Element3D, Element3D > replacementFunction)

  Replaces each element in the scene with the element returned by the replacementFunction .

Parameters

replacementFunction	*A function replacing each Element3D in the scene with another Element3D.*

- void Replace (Func< Element3D, IEnumerable< Element3D > > replacementFunction)

  Replaces each element in the scene with the element(s) returned by the replacementFunction .

Parameters

replacementFunction	*A function replacing each Element3D in the scene with 0 or more Element3Ds.*

Properties

- object SceneLock [get]

  An object used to synchronise multithreaded rendering of the same scene.
- IEnumerable< Element3D > SceneElements [get]

  The Element3Ds constituting the scene.

7.161.1 Detailed Description

Represents a 3D scene.

Definition at line 66 of file Scene.cs.

7.161.2 Constructor & Destructor Documentation

7.161.2.1 Scene()

```
VectSharp.ThreeD.Scene.Scene ( )
```

Creates a new Scene.

Definition at line 79 of file Scene.cs.

7.161.3 Member Function Documentation

7.161.3.1 AddElement()

```
void VectSharp.ThreeD.Scene.AddElement (
            Element3D element )
```

Adds the specified *element* to the scene.

Parameters

element	The Element3D to add.

Implements VectSharp.ThreeD.IScene.

Definition at line 86 of file Scene.cs.

7.161.3.2 AddRange()

```
void VectSharp.ThreeD.Scene.AddRange (
            IEnumerable< Element3D > elements )
```

Adds the specified *elements* to the scene.

Parameters

elements	A collection of Element3Ds to add.

Implements VectSharp.ThreeD.IScene.

Definition at line 92 of file Scene.cs.

7.161.3.3 Replace() [1/2]

```
void VectSharp.ThreeD.Scene.Replace (
            Func< Element3D, Element3D > replacementFunction )
```

Replaces each element in the scene with the element returned by the *replacementFunction* .

Parameters

replacementFunction	A function replacing each Element3D in the scene with another Element3D.

Implements VectSharp.ThreeD.IScene.

Definition at line 98 of file Scene.cs.

7.161.3.4 Replace() [2/2]

```
void VectSharp.ThreeD.Scene.Replace (
            Func< Element3D, IEnumerable< Element3D > > replacementFunction )
```

Replaces each element in the scene with the element(s) returned by the *replacementFunction* .

Parameters

replacementFunction	A function replacing each Element3D in the scene with 0 or more Element3Ds.

Implements VectSharp.ThreeD.IScene.

Definition at line 107 of file Scene.cs.

7.161.4 Property Documentation

7.161.4.1 SceneElements

```
IEnumerable<Element3D> VectSharp.ThreeD.Scene.SceneElements  [get]
```

The Element3Ds constituting the scene.

Implements VectSharp.ThreeD.IScene.

Definition at line 74 of file Scene.cs.

7.161.4.2 SceneLock

```
object VectSharp.ThreeD.Scene.SceneLock  [get]
```

An object used to synchronise multithreaded rendering of the same scene.

Implements VectSharp.ThreeD.IScene.

Definition at line 69 of file Scene.cs.

The documentation for this class was generated from the following file:

- VectSharp.ThreeD/Scene.cs

7.162 VectSharp.Segment Class Reference

Represents a segment as part of a GraphicsPath.

Public Member Functions

- abstract Segment Clone ()

  Creates a copy of the Segment.
- abstract double Measure (Point previousPoint)

  Computes the length of the Segment.
- abstract Point GetPointAt (Point previousPoint, double position)

  Gets the point on the Segment at the specified (relative) position).
- abstract Point GetTangentAt (Point previousPoint, double position)

  Gets the tangent to the Segment at the specified (relative) position).
- abstract IEnumerable< Segment > Linearise (Point? previousPoint, double resolution)

  Transform the segment into a series of linear segments. Segments that are already linear are not changed.
- abstract IEnumerable< Point > GetLinearisationTangents (Point? previousPoint, double resolution)

  Gets the tanget at the points at which the segment would be linearised.
- abstract IEnumerable< Segment > Transform (Func< Point, Point > transformationFunction)

  Applies an arbitrary transformation to all of the points of the Segment.
- abstract IEnumerable< Segment > Flatten (Point? previousPoint, double flatness)

  Flattens the Segment, replacing curve segments with series of line segments that approximate them, ensuring the specified maximum deviation from the original path.
- abstract IEnumerable<(Point point, Point tangent)> FlattenForOffsetAndGetTangents (Point? previousPoint, double offset, double flatness)

  Flattens the Segment, replacing curve segments with series of line segments that approximate them, ensuring the specified maximum deviation from the original path, assuming that the Segment will be drawn with the specified offset .

Properties

- abstract SegmentType Type [get]

  The type of the Segment.
- Point[] Points [get]

  The points used to define the Segment.
- virtual Point Point [get]

  The end point of the Segment.

7.162.1 Detailed Description

Represents a segment as part of a GraphicsPath.

Definition at line 29 of file Segment.cs.

7.162.2 Member Function Documentation

7.162.2.1 Clone()

```
abstract Segment VectSharp.Segment.Clone ( )  [pure virtual]
```

Creates a copy of the Segment.

Returns

A copy of the Segment.

7.162.2.2 Flatten()

```
abstract IEnumerable< Segment > VectSharp.Segment.Flatten (
            Point?  previousPoint,
            double flatness )  [pure virtual]
```

Flattens the Segment, replacing curve segments with series of line segments that approximate them, ensuring the specified maximum deviation from the original path.

Parameters

previousPoint	The point from which the Segment starts (i.e. the endpoint of the previous Segment).
flatness	The maximum deviation from the original path.

Returns

A collection of Segments composed only of linear segments that approximates the current Segment.

7.162.2.3 FlattenForOffsetAndGetTangents()

```
abstract IEnumerable<(Point point, Point tangent)> VectSharp.Segment.FlattenForOffsetAndGet↩
Tangents (
            Point?  previousPoint,
            double offset,
            double flatness )  [pure virtual]
```

Flattens the Segment, replacing curve segments with series of line segments that approximate them, ensuring the specified maximum deviation from the original path, assuming that the Segment will be drawn with the specified *offset* .

Parameters

previousPoint	The point from which the Segment starts (i.e. the endpoint of the previous Segment).
offset	The offset that will be used to draw the Segment (e.g., the line width of the stroke).
flatness	The maximum deviation from the original path.

Returns

A collection of tuples where the first element is a Point representing the end-point of a linear segment, and the second element is a Point containing the value of the tangent to the original Segment at that point.

7.162.2.4 GetLinearisationTangents()

```
abstract IEnumerable< Point > VectSharp.Segment.GetLinearisationTangents (
            Point?  previousPoint,
            double resolution )  [pure virtual]
```

Gets the tanget at the points at which the segment would be linearised.

Parameters

previousPoint	The point from which the Segment starts (i.e. the endpoint of the previous Segment).
resolution	The absolute length between successive samples in curve segments.

Returns

A collection of tangents at the points in which the segment would be linearised.

7.162.2.5 GetPointAt()

```
abstract Point VectSharp.Segment.GetPointAt (
            Point previousPoint,
            double position )  [pure virtual]
```

Gets the point on the Segment at the specified (relative) *position*).

Parameters

previousPoint	The point from which the Segment starts (i.e. the endpoint of the previous Segment).
position	The relative position on the Segment (0 is the start of the Segment, 1 is the end of the Segment).

Returns

The point at the specified position.

7.162.2.6 GetTangentAt()

```
abstract Point VectSharp.Segment.GetTangentAt (
            Point previousPoint,
            double position )  [pure virtual]
```

Gets the tangent to the Segment at the specified (relative) *position*).

Parameters

previousPoint	The point from which the Segment starts (i.e. the endpoint of the previous Segment).
position	The relative position on the Segment (0 is the start of the Segment, 1 is the end of the Segment).

Returns

The tangent to the point at the specified position.

7.162.2.7 Linearise()

```
abstract IEnumerable< Segment > VectSharp.Segment.Linearise (
            Point?  previousPoint,
            double resolution )  [pure virtual]
```

Transform the segment into a series of linear segments. Segments that are already linear are not changed.

Parameters

previousPoint	The point from which the Segment starts (i.e. the endpoint of the previous Segment).
resolution	The absolute length between successive samples in curve segments.

Returns

A collection of linear segments that approximate the current segment.

7.162.2.8 Measure()

```
abstract double VectSharp.Segment.Measure (
            Point previousPoint )  [pure virtual]
```

Computes the length of the Segment.

Parameters

previousPoint	The point from which the Segment starts (i.e. the endpoint of the previous Segment).

Returns

The length of the segment.

7.162.2.9 Transform()

```
abstract IEnumerable< Segment > VectSharp.Segment.Transform (
            Func< Point, Point > transformationFunction )  [pure virtual]
```

Applies an arbitrary transformation to all of the points of the Segment.

Parameters

transformationFunction	An arbitrary transformation function.

Returns

A collection of Segments that have been transformed according to the *transformationFunction* .

7.162.3 Property Documentation

7.162.3.1 Point

```
virtual Point VectSharp.Segment.Point  [get]
```

The end point of the Segment.

Definition at line 45 of file Segment.cs.

7.162.3.2 Points

```
Point [] VectSharp.Segment.Points  [get]
```

The points used to define the Segment.

Definition at line 40 of file Segment.cs.

7.162.3.3 Type

```
abstract SegmentType VectSharp.Segment.Type  [get]
```

The type of the Segment.

Definition at line 35 of file Segment.cs.

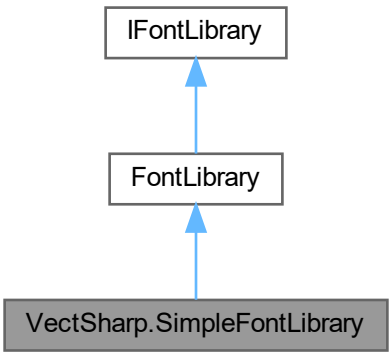
The documentation for this class was generated from the following file:

- VectSharp/Segment.cs

7.163 VectSharp.SimpleFontLibrary Class Reference

A font library that can be used to cache and resolve font family names.

Inheritance diagram for VectSharp.SimpleFontLibrary:



Public Member Functions

- SimpleFontLibrary (IFontLibrary standardFontLibrary)

  Create a new SimpleFontLibrary instance.
- SimpleFontLibrary ()

  Create a new SimpleFontLibrary instance, using the default font library to resolve the standard font families.
- SimpleFontLibrary (FontFamily timesRoman, FontFamily timesBold, FontFamily timesItalic, FontFamily timesBoldItalic, FontFamily helvetica, FontFamily helveticaBold, FontFamily helveticaOblique, FontFamily helveticaBoldOblique, FontFamily courier, FontFamily courierBold, FontFamily courierOblique, FontFamily courierBoldOblique, FontFamily symbol, FontFamily zapfdingbats)

  Create a new SimpleFontLibrary instance, with the specified replacements for the standard font families.
- SimpleFontLibrary (string timesRoman, string timesBold, string timesItalic, string timesBoldItalic, string helvetica, string helveticaBold, string helveticaOblique, string helveticaBoldOblique, string courier, string courierBold, string courierOblique, string courierBoldOblique, string symbol, string zapfdingbats)

  Create a new SimpleFontLibrary instance, with the specified replacements for the standard font families.
- void Add (string fontFamilyName, FontFamily fontFamily)

  Add the specified font family to the library with the specified name.
- void Add (FontFamily fontFamily)

  Add the specified font family to the library.
- void Add (string fileName)

  Add the font family contained in the specified True Type Font file to the library.
- void Add (string fontFamily, string fileName)

  Add the font family contained in the specified True Type Font file to the library, with the specified name. The font family is not loaded until it is requested for the first time.
- override FontFamily ResolveFontFamily (FontFamily.StandardFontFamilies standardFontFamily)

  Create a new font family from the specified standard font family name.

Parameters

standardFontFamily	*The standard name of the font family.*

Returns

A FontFamily object corresponding to the specified font family.

- override FontFamily ResolveFontFamily (string fontFamily)

  Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised.

Parameters

fontFamily	*The name of the font family to create, or the path to a TTF file.*

Returns

If the font family name or the true type file is valid, a FontFamily object corresponding to the specified font family.

7.163.1 Detailed Description

A font library that can be used to cache and resolve font family names.

Definition at line 185 of file FontLibrary.cs.

7.163.2 Constructor & Destructor Documentation

7.163.2.1 SimpleFontLibrary() [1/4]

```
VectSharp.SimpleFontLibrary.SimpleFontLibrary (
            IFontLibrary standardFontLibrary )
```

Create a new SimpleFontLibrary instance.

Parameters

standardFontLibrary	An existing font library that will be used to resolve the standard font families.

Definition at line 197 of file FontLibrary.cs.

7.163.2.2 SimpleFontLibrary() [2/4]

```
VectSharp.SimpleFontLibrary.SimpleFontLibrary ( )
```

Create a new SimpleFontLibrary instance, using the default font library to resolve the standard font families.

Definition at line 217 of file FontLibrary.cs.

7.163.2.3 SimpleFontLibrary() [3/4]

```
VectSharp.SimpleFontLibrary.SimpleFontLibrary (
            FontFamily timesRoman,
            FontFamily timesBold,
            FontFamily timesItalic,
            FontFamily timesBoldItalic,
            FontFamily helvetica,
            FontFamily helveticaBold,
            FontFamily helveticaOblique,
            FontFamily helveticaBoldOblique,
            FontFamily courier,
            FontFamily courierBold,
            FontFamily courierOblique,
            FontFamily courierBoldOblique,
            FontFamily symbol,
            FontFamily zapfdingbats )
```

Create a new SimpleFontLibrary instance, with the specified replacements for the standard font families.

Parameters

timesRoman	The font family to use for the Times-Roman standard font.
timesBold	The font family to use for the Times-Bold standard font.
timesItalic	The font family to use for the Times-Italic standard font.
timesBoldItalic	The font family to use for the Times-BoldItalic standard font.
helvetica	The font family to use for the Helvetica standard font.

Parameters

helveticaBold	The font family to use for the Helvetica-Bold standard font.
helveticaOblique	The font family to use for the Helvetica-Oblique standard font.
helveticaBoldOblique	The font family to use for the Helvetica-BoldOblique standard font.
courier	The font family to use for the Courier standard font.
courierBold	The font family to use for the Courier-Bold standard font.
courierOblique	The font family to use for the Courier-Oblique standard font.
courierBoldOblique	The font family to use for the Courier-BoldOblique standard font.
symbol	The font family to use for the Symbol standard font.
zapfdingbats	The font family to use for the Zapfdingbats standard font.

Definition at line 239 of file FontLibrary.cs.

7.163.2.4 SimpleFontLibrary() [4/4]

```
VectSharp.SimpleFontLibrary.SimpleFontLibrary (
            string timesRoman,
            string timesBold,
            string timesItalic,
            string timesBoldItalic,
            string helvetica,
            string helveticaBold,
            string helveticaOblique,
            string helveticaBoldOblique,
            string courier,
            string courierBold,
            string courierOblique,
            string courierBoldOblique,
            string symbol,
            string zapfdingbats )
```

Create a new SimpleFontLibrary instance, with the specified replacements for the standard font families.

Parameters

timesRoman	The font family to use for the Times-Roman standard font.
timesBold	The font family to use for the Times-Bold standard font.
timesItalic	The font family to use for the Times-Italic standard font.
timesBoldItalic	The font family to use for the Times-BoldItalic standard font.
helvetica	The font family to use for the Helvetica standard font.
helveticaBold	The font family to use for the Helvetica-Bold standard font.
helveticaOblique	The font family to use for the Helvetica-Oblique standard font.
helveticaBoldOblique	The font family to use for the Helvetica-BoldOblique standard font.
courier	The font family to use for the Courier standard font.
courierBold	The font family to use for the Courier-Bold standard font.
courierOblique	The font family to use for the Courier-Oblique standard font.
courierBoldOblique	The font family to use for the Courier-BoldOblique standard font.
symbol	The font family to use for the Symbol standard font.
zapfdingbats	The font family to use for the Zapfdingbats standard font.

Definition at line 297 of file FontLibrary.cs.

7.163.3 Member Function Documentation

7.163.3.1 Add() [1/4]

```
void VectSharp.SimpleFontLibrary.Add (
            FontFamily fontFamily )
```

Add the specified font family to the library.

Parameters

fontFamily	The font family to add.

Definition at line 360 of file FontLibrary.cs.

7.163.3.2 Add() [2/4]

```
void VectSharp.SimpleFontLibrary.Add (
            string fileName )
```

Add the font family contained in the specified True Type Font file to the library.

Parameters

fileName	The path to the TTF file containing the font family.

Definition at line 372 of file FontLibrary.cs.

7.163.3.3 Add() [3/4]

```
void VectSharp.SimpleFontLibrary.Add (
            string fontFamily,
            string fileName )
```

Add the font family contained in the specified True Type Font file to the library, with the specified name. The font family is not loaded until it is requested for the first time.

Parameters

fontFamily	The name of the font family.
fileName	The path to the TTF file containing the font family.

Definition at line 387 of file FontLibrary.cs.

7.163.3.4 Add() [4/4]

```
void VectSharp.SimpleFontLibrary.Add (
            string fontFamilyName,
            FontFamily fontFamily )
```

Add the specified font family to the library with the specified name.

Parameters

fontFamilyName	The name of the font family.
fontFamily	The font family to add.

Definition at line 343 of file FontLibrary.cs.

7.163.3.5 ResolveFontFamily() [1/2]

```
override FontFamily VectSharp.SimpleFontLibrary.ResolveFontFamily (
            FontFamily.StandardFontFamilies standardFontFamily )  [virtual]
```

Create a new font family from the specified standard font family name.

Parameters

standardFontFamily	The standard name of the font family.

Returns

A FontFamily object corresponding to the specified font family.

Implements VectSharp.FontLibrary.

Definition at line 394 of file FontLibrary.cs.

7.163.3.6 ResolveFontFamily() [2/2]

```
override FontFamily VectSharp.SimpleFontLibrary.ResolveFontFamily (
            string fontFamily )  [virtual]
```

Create a new font family from the specified family name or true type file. If the family name or the true type file are not valid, an exception might be raised.

Parameters

fontFamily	The name of the font family to create, or the path to a TTF file.

Returns

If the font family name or the true type file is valid, a FontFamily object corresponding to the specified font family.

Implements VectSharp.FontLibrary.

Definition at line 400 of file FontLibrary.cs.

The documentation for this class was generated from the following file:

- VectSharp/FontLibrary.cs

7.164 VectSharp.Size Struct Reference

Represents the size of an object.

Public Member Functions

- Size (double width, double height)

  Create a new Size.

Public Attributes

- double Width

  Width of the object.
- double Height

  Height of the object.

7.164.1 Detailed Description

Represents the size of an object.

Definition at line 273 of file Point.cs.

7.164.2 Constructor & Destructor Documentation

7.164.2.1 Size()

```
VectSharp.Size.Size (
            double width,
            double height )
```

Create a new Size.

Parameters

width	The width of the object.
height	The height of the object.

Definition at line 290 of file Point.cs.

7.164.3 Member Data Documentation

7.164.3.1 Height

```
double VectSharp.Size.Height
```

Height of the object.

Definition at line 283 of file Point.cs.

7.164.3.2 Width

```
double VectSharp.Size.Width
```

Width of the object.

Definition at line 278 of file Point.cs.

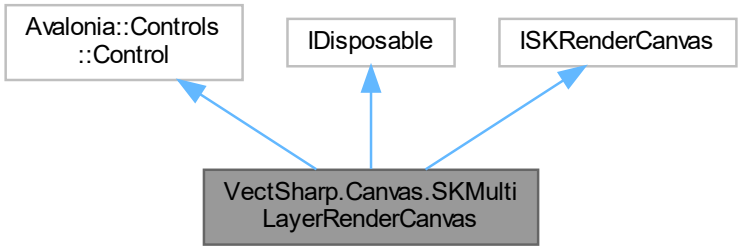
The documentation for this struct was generated from the following file:

- VectSharp/Point.cs

7.165 VectSharp.Canvas.SKMultiLayerRenderCanvas Class Reference

Represents a multi-threaded, triple-buffered canvas on which the image is drawn using SkiaSharp.

Inheritance diagram for VectSharp.Canvas.SKMultiLayerRenderCanvas:



Public Member Functions

- SKMultiLayerRenderCanvas (Document document, Colour backgroundColour, double width, double height, List< SKRenderAction > layerTransforms=null)

  Create a new SKMultiLayerRenderCanvas from a Document, where each page represents a layer.
- SKMultiLayerRenderCanvas (IEnumerable< Page > layers, Colour backgroundColour, double width, double height, List< SKRenderAction > layerTransforms=null)

  Create a new SKMultiLayerRenderCanvas from a collection of Pages, each representing a layer.
- SKMultiLayerRenderCanvas (List< SKRenderContext > contents, List< SKRenderAction > content↩ Transforms, Colour backgroundColour, double width, double height)

  Create a new SKMultiLayerRenderCanvas from a list of SKRenderContexts, each representing a layer.
- void UpdateWith (List< SKRenderContext > contents, List< SKRenderAction > contentTransforms, Colour backgroundColour, double width, double height)

  Replace the contents of the SKMultiLayerRenderCanvas with the specified layers.
- void UpdateLayer (int layer, SKRenderContext newContent, SKRenderAction newTransform)

  Replace a single layer with the specified content.
- void AddLayer (SKRenderContext newContent, SKRenderAction newTransform)

  Add a new layer to the image.
- void InsertLayer (int index, SKRenderContext newContent, SKRenderAction newTransform)

  Insert a new layer at the specified index.
- void RemoveLayer (int layer)

  Remove the specified layer from the image.
- void SwitchLayers (int layer1, int layer2)

  Switch the position of the two specified layers.
- void MoveLayer (int oldIndex, int newIndex)

  Move the specified layer to the specified position, shifting all other layers as necessary.
- RenderTargetBitmap RenderAtResolution (int width, int height, SKColor? background=null)

  Render the image to a bitmap at the specified resolution.
- void InvalidateDirty ()

  Invalidate the contents of the canvas, forcing it to redraw itself.
- void InvalidateZIndex ()

  Invalidate the contents of the canvas, specifying that the order of the layers has changed.
- override void Render (DrawingContext context)
- void Dispose ()

Public Attributes

- List< List< SKRenderAction > > RenderActions

  The list of render actions. Each element in this list is itself a list, containing the actions that correspond to a layer in the image.
- List< SKRenderAction > LayerTransforms

  The list of transforms associated with each layer.
- object RenderLock = new object()

  A lock for the rendering loop. The public methods of this class already lock on this, but you may need it if you want to directly manipulate the contents of the canvas.

Properties

- double PageWidth [get, set]

  The width of the page that is rendered on this canvas.
- double PageHeight [get, set]

  The height of the page that is rendered on this canvas.
- RelativePoint ClipMargin = new RelativePoint(0, 0, RelativeUnit.Absolute) [get, set]

  Defines an overdraw margin for the clipping area. Set this to a larger value if quickly moving the canvas around (e.g. in a ZoomBorder) causes the edge of the image to disappear for a few milliseconds and that annoys you. If RelativePoint.Unit is set to RelativeUnit.Absolute, the value corresponds to units on the VectSharp.Page; if it is set to RelativeUnit.Relative, it corresponds to a proportion of the visible area.

7.165.1 Detailed Description

Represents a multi-threaded, triple-buffered canvas on which the image is drawn using SkiaSharp.

Definition at line 39 of file SKMultiLayerRenderCanvas.cs.

7.165.2 Constructor & Destructor Documentation

7.165.2.1 SKMultiLayerRenderCanvas() [1/3]

```
VectSharp.Canvas.SKMultiLayerRenderCanvas.SKMultiLayerRenderCanvas (
            Document document,
            Colour backgroundColour,
            double width,
            double height,
            List< SKRenderAction > layerTransforms = null )
```

Create a new SKMultiLayerRenderCanvas from a Document, where each page represents a layer.

Parameters

document	The document containing the layers as Pages.
layerTransforms	A list of transforms associated with each layer. This list should contain the same number of elements as the number of pages in *document* . This is useful to manipulate the position of each layer individually. If this is null, an identity transform is applied to each layer.
backgroundColour	The background colour of the canvas.
width	The width of the canvas and the pages it contains.
height	The height of the canvas and the pages it contains.

Definition at line 82 of file SKMultiLayerRenderCanvas.cs.

7.165.2.2 SKMultiLayerRenderCanvas() [2/3]

```
VectSharp.Canvas.SKMultiLayerRenderCanvas.SKMultiLayerRenderCanvas (
            IEnumerable< Page > layers,
            Colour backgroundColour,
            double width,
            double height,
            List< SKRenderAction > layerTransforms = null )
```

Create a new SKMultiLayerRenderCanvas from a collection of Pages, each representing a layer.

Parameters

layers	The contents of the canvas. Each element in this list represents a layer.
layerTransforms	A list of transforms associated with each layer. This list should contain the same number of elements as *layers* . This is useful to manipulate the position of each layer individually. If this is null, an identity transform is applied to each layer.
backgroundColour	The background colour of the canvas.
width	The width of the canvas and the pages it contains.
height	The height of the canvas and the pages it contains.

Definition at line 92 of file SKMultiLayerRenderCanvas.cs.

7.165.2.3 SKMultiLayerRenderCanvas() [3/3]

```
VectSharp.Canvas.SKMultiLayerRenderCanvas.SKMultiLayerRenderCanvas (
            List< SKRenderContext > contents,
            List< SKRenderAction > contentTransforms,
            Colour backgroundColour,
            double width,
            double height )
```

Create a new SKMultiLayerRenderCanvas from a list of SKRenderContexts, each representing a layer.

Parameters

contents	The contents of the canvas. Each element in this list represents a layer. A Page can be converded to a SKRenderContext through the CopyToSKRenderContext method.
contentTransforms	A list of transforms associated with each layer. This list should contain the same number of elements as *contents* . This is useful to manipulate the position of each layer individually.
backgroundColour	The background colour of the canvas.
width	The width of the canvas and the page it contains.
height	The height of the canvas and the page it contains.

Definition at line 125 of file SKMultiLayerRenderCanvas.cs.

7.165.3 Member Function Documentation

7.165.3.1 AddLayer()

```
void VectSharp.Canvas.SKMultiLayerRenderCanvas.AddLayer (
            SKRenderContext newContent,
            SKRenderAction newTransform )
```

Add a new layer to the image.

Parameters

newContent	The contents of the new layer. A Page can be converded to a SKRenderContext through the CopyToSKRenderContext method.
newTransform	The transform for the new layer.

Definition at line 256 of file SKMultiLayerRenderCanvas.cs.

7.165.3.2 Dispose()

```
void VectSharp.Canvas.SKMultiLayerRenderCanvas.Dispose ( )
```

Definition at line 1134 of file SKMultiLayerRenderCanvas.cs.

7.165.3.3 InsertLayer()

```
void VectSharp.Canvas.SKMultiLayerRenderCanvas.InsertLayer (
            int index,
            SKRenderContext newContent,
            SKRenderAction newTransform )
```

Insert a new layer at the specified index.

Parameters

index	The position at which the new layer will be inserted.
newContent	The contents of the new layer.
newTransform	The transform for the new layer.

Definition at line 289 of file SKMultiLayerRenderCanvas.cs.

7.165.3.4 InvalidateDirty()

```
void VectSharp.Canvas.SKMultiLayerRenderCanvas.InvalidateDirty ( )
```

Invalidate the contents of the canvas, forcing it to redraw itself.

Definition at line 943 of file SKMultiLayerRenderCanvas.cs.

7.165.3.5 InvalidateZIndex()

```
void VectSharp.Canvas.SKMultiLayerRenderCanvas.InvalidateZIndex ( )
```

Invalidate the contents of the canvas, specifying that the order of the layers has changed.

Definition at line 952 of file SKMultiLayerRenderCanvas.cs.

7.165.3.6 MoveLayer()

```
void VectSharp.Canvas.SKMultiLayerRenderCanvas.MoveLayer (
            int oldIndex,
            int newIndex )
```

Move the specified layer to the specified position, shifting all other layers as necessary.

Parameters

oldIndex	The current index of the layer to move.
newIndex	The final index of the layer. Layers after this will be shifted by 1 in order to accommodate the moved layer.

Definition at line 369 of file SKMultiLayerRenderCanvas.cs.

7.165.3.7 RemoveLayer()

```
void VectSharp.Canvas.SKMultiLayerRenderCanvas.RemoveLayer (
            int layer )
```

Remove the specified layer from the image.

Parameters

layer	The index of the layer to remove.

Definition at line 320 of file SKMultiLayerRenderCanvas.cs.

7.165.3.8 Render()

```
override void VectSharp.Canvas.SKMultiLayerRenderCanvas.Render (
            DrawingContext context )
```

Definition at line 977 of file SKMultiLayerRenderCanvas.cs.

7.165.3.9 RenderAtResolution()

```
RenderTargetBitmap VectSharp.Canvas.SKMultiLayerRenderCanvas.RenderAtResolution (
            int width,
            int height,
            SKColor?  background = null )
```

Render the image to a bitmap at the specified resolution.

Parameters

width	The width of the rendered image. Note that the actual width of the returned image might be lower than this, depending on the aspect ratio of the image.
height	The height of the rendered image. Note that the actual height of the returned image might be lower than this, depending on the aspect ratio of the image.
background	The background colour for the image. If this is `null`, the current background colour is used.

Returns

A RenderTargetBitmap containing the image rendered at the specified resolution.

Definition at line 705 of file SKMultiLayerRenderCanvas.cs.

7.165.3.10 SwitchLayers()

```
void VectSharp.Canvas.SKMultiLayerRenderCanvas.SwitchLayers (
            int layer1,
            int layer2 )
```

Switch the position of the two specified layers.

Parameters

layer1	The index of the first layer to switch.
layer2	The index of the second layer to switch.

Definition at line 344 of file SKMultiLayerRenderCanvas.cs.

7.165.3.11 UpdateLayer()

```
void VectSharp.Canvas.SKMultiLayerRenderCanvas.UpdateLayer (
            int layer,
            SKRenderContext newContent,
            SKRenderAction newTransform )
```

Replace a single layer with the specified content.

Parameters

layer	The index of the layer to replace.
newContent	The new contents of the layer. A Page can be converded to a SKRenderContext through the CopyToSKRenderContext method.
newTransform	The new transform for the layer.

Definition at line 224 of file SKMultiLayerRenderCanvas.cs.

7.165.3.12 UpdateWith()

```
void VectSharp.Canvas.SKMultiLayerRenderCanvas.UpdateWith (
            List< SKRenderContext > contents,
            List< SKRenderAction > contentTransforms,
            Colour backgroundColour,
            double width,
            double height )
```

Replace the contents of the SKMultiLayerRenderCanvas with the specified layers.

Parameters

contents	The contents of the canvas. Each element in this list represents a layer. A Page can be converded to a SKRenderContext through the CopyToSKRenderContext method.
contentTransforms	A list of transforms associated with each layer. This list should contain the same number of elements as *contents* . This is useful to manipulate the position of each layer individually.
backgroundColour	The background colour of the canvas.
width	The width of the canvas and the page it contains.
height	The height of the canvas and the page it contains.

Definition at line 146 of file SKMultiLayerRenderCanvas.cs.

7.165.4 Member Data Documentation

7.165.4.1 LayerTransforms

```
List<SKRenderAction> VectSharp.Canvas.SKMultiLayerRenderCanvas.LayerTransforms
```

The list of transforms associated with each layer.

Definition at line 69 of file SKMultiLayerRenderCanvas.cs.

7.165.4.2 RenderActions

```
List<List<SKRenderAction> > VectSharp.Canvas.SKMultiLayerRenderCanvas.RenderActions
```

The list of render actions. Each element in this list is itself a list, containing the actions that correspond to a layer in the image.

Definition at line 64 of file SKMultiLayerRenderCanvas.cs.

7.165.4.3 RenderLock

```
object VectSharp.Canvas.SKMultiLayerRenderCanvas.RenderLock = new object()
```

A lock for the rendering loop. The public methods of this class already lock on this, but you may need it if you want to directly manipulate the contents of the canvas.

Definition at line 696 of file SKMultiLayerRenderCanvas.cs.

7.165.5 Property Documentation

7.165.5.1 ClipMargin

```
RelativePoint VectSharp.Canvas.SKMultiLayerRenderCanvas.ClipMargin = new RelativePoint(0, 0,
RelativeUnit.Absolute)  [get], [set]
```

Defines an overdraw margin for the clipping area. Set this to a larger value if quickly moving the canvas around (e.g. in a ZoomBorder) causes the edge of the image to disappear for a few milliseconds and that annoys you. If RelativePoint.Unit is set to RelativeUnit.Absolute, the value corresponds to units on the VectSharp.Page; if it is set to RelativeUnit.Relative, it corresponds to a proportion of the visible area.

Definition at line 59 of file SKMultiLayerRenderCanvas.cs.

7.165.5.2 PageHeight

```
double VectSharp.Canvas.SKMultiLayerRenderCanvas.PageHeight  [get], [set]
```

The height of the page that is rendered on this canvas.

Definition at line 49 of file SKMultiLayerRenderCanvas.cs.

7.165.5.3 PageWidth

```
double VectSharp.Canvas.SKMultiLayerRenderCanvas.PageWidth  [get], [set]
```

The width of the page that is rendered on this canvas.

Definition at line 44 of file SKMultiLayerRenderCanvas.cs.

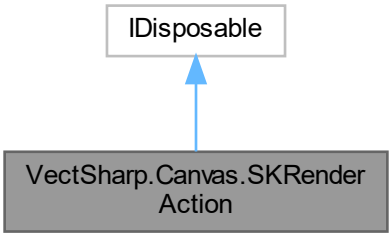
The documentation for this class was generated from the following file:

- VectSharp.Canvas/SKMultiLayerRenderCanvas.cs

7.166 VectSharp.Canvas.SKRenderAction Class Reference

Represents a light-weight rendering action.

Inheritance diagram for VectSharp.Canvas.SKRenderAction:



Public Types

- enum ActionTypes

  Types of rendering actions.

Public Member Functions

- void InvalidateHitTestPath ()
- void InvalidateVisual ()
- void InvalidateZIndex ()
- void InvalidateAll ()
- void Dispose ()

Static Public Member Functions

- static SKRenderAction PathAction (SKPath path, SKPaint paint, string tag=null)

  Creates a new SKRenderAction representing a path.
- static SKRenderAction ClipAction (SKPath clippingPath, string tag=null)

  Creates a new SKRenderAction representing a clipping action.
- static SKRenderAction TextAction (string text, float x, float y, SKFont font, SKPaint paint, string tag=null)

  Creates a new SKRenderAction representing text.
- static SKRenderAction ImageAction (string imageId, SKRect sourceRect, SKRect destinationRect, string tag=null)

  Creates a new SKRenderAction representing an image.
- static SKRenderAction TransformAction (SKMatrix transform, string tag=null)

  Creates a new SKRenderAction representing a transform.
- static SKRenderAction SaveAction (string tag=null)

  Creates a new SKRenderAction that saves the current graphics state.
- static SKRenderAction RestoreAction (string tag=null)

  Creates a new SKRenderAction that saves the current graphics state.
- static SKRenderAction DrawFilteredGraphicsAction (SKRenderContext graphics, IFilter filter, string tag=null)

  Create a new SKRenderAction that draws some graphics with a filter.

Properties

- bool Disposed `[get]`

  Returns a boolean value indicating whether the current instance has been disposed.
- ActionTypes ActionType `[get]`

  Type of the rendering action.
- SKPath Path `[get, set]`

  Path that needs to be rendered (null if the action type is not ActionTypes.Path). If you change this, you probably want to call this object's InvalidateHitTestPath method.
- string Text `[get, set]`

  Text that needs to be rendered (null if the action type is not ActionTypes.Text). If you change this, you probably want to call this object's InvalidateHitTestPath method.
- SKFont Font `[get, set]`

  The font that will be used to render the text (null if the action type is not ActionTypes.Text). If you change this, you probably want to call this object's InvalidateHitTestPath method.
- float TextX `[get, set]`

  The X coordainate at which the text will be drawn (null if the action type is not ActionTypes.Text). If you change this, you probably want to call this object's InvalidateHitTestPath method.
- float TextY `[get, set]`

  The Y coordainate at which the text will be drawn (null if the action type is not ActionTypes.Text). If you change this, you probably want to call this object's InvalidateHitTestPath method.
- SKPaint Paint `[get, set]`

Paint used to render the text or path (`null` if the action type is neither ActionTypes.Text nor ActionTypes.Path). If you change this, you probably want to call this object's InvalidateHitTestPath method.

- string ImageId `[get, set]`

  Univocal identifier of the image that needs to be drawn.

- SKRect? ImageSource `[get, set]`

  The source rectangle of the image (`null` if the action type is not ActionTypes.RasterImage). If you change this, you probably want to call this object's InvalidateVisual method.

- SKRect? ImageDestination `[get, set]`

  The destination rectangle of the image (`null` if the action type is not ActionTypes.RasterImage). If you change this, you probably want to call this object's InvalidateHitTestPath method.

- SKMatrix? Transform = null `[get, set]`

  The transformation matrix that will be applied to the current coordinate system (`null` if the action type is not ActionTypes.Transform). If you change this, you probably want to call this object's InvalidateVisual method.

- string Tag `[get]`

  A tag to access the SKRenderAction.

- uint ZIndex = 0 `[get, set]`

  The Z-index of the rendering action (an action with a higher Z-index will always appear above an action with a lower Z-index). The more different values there are for the Z-index, the slower the rendering, so keep use of this property to a minimum. If you change this, you probably want to call this object's InvalidateZIndex method.

- SKRenderContext Graphics = null `[get, set]`

  The graphics that will be drawn with the specified Filter. If you change this, you probably want to call this object's InvalidateVisual method.

- IFilter Filter `[get, set]`

  The filter with which the Graphics is drawn. If you change this, you probably want to call this object's InvalidateVisual method.

- object Payload `[get, set]`

  An arbitrary object associated with the RenderAction.

- Avalonia.Controls.Canvas Parent `[get]`

  The container of this SKRenderAction.

Events

- EventHandler< Avalonia.Input.PointerEventArgs > PointerEntered

  Raised when the pointer enters the area covered by the SKRenderAction.

- EventHandler< Avalonia.Input.PointerEventArgs > PointerExited

  Raised when the pointer leaves the area covered by the SKRenderAction.

- EventHandler< Avalonia.Input.PointerPressedEventArgs > PointerPressed

  Raised when the pointer is pressed while over the area covered by the SKRenderAction.

- EventHandler< Avalonia.Input.PointerReleasedEventArgs > PointerReleased

  Raised when the pointer is released after a PointerPressed event.

7.166.1 Detailed Description

Represents a light-weight rendering action.

Definition at line 30 of file SKRenderContext.cs.

7.166.2 Member Enumeration Documentation

7.166.2.1 ActionTypes

```
enum VectSharp.Canvas.SKRenderAction.ActionTypes
```

Types of rendering actions.

Definition at line 42 of file SKRenderContext.cs.

7.166.3 Member Function Documentation

7.166.3.1 ClipAction()

```
static SKRenderAction VectSharp.Canvas.SKRenderAction.ClipAction (
            SKPath clippingPath,
            string tag = null )  [static]
```

Creates a new SKRenderAction representing a clipping action.

Parameters

clippingPath	The path to be used for clipping.
tag	A tag to access the SKRenderAction.

Returns

A new SKRenderAction representing a clipping action.

Definition at line 345 of file SKRenderContext.cs.

7.166.3.2 Dispose()

```
void VectSharp.Canvas.SKRenderAction.Dispose ( )
```

Definition at line 494 of file SKRenderContext.cs.

7.166.3.3 DrawFilteredGraphicsAction()

```
static SKRenderAction VectSharp.Canvas.SKRenderAction.DrawFilteredGraphicsAction (
            SKRenderContext graphics,
            IFilter filter,
            string tag = null )  [static]
```

Create a new SKRenderAction that draws some graphics with a filter.

Returns

A new SKRenderAction that draws some graphics with a filter.

Definition at line 464 of file SKRenderContext.cs.

7.166.3.4 ImageAction()

```
static SKRenderAction VectSharp.Canvas.SKRenderAction.ImageAction (
            string imageId,
            SKRect sourceRect,
            SKRect destinationRect,
            string tag = null )  [static]
```

Creates a new SKRenderAction representing an image.

Parameters

imageId	The univocal identifier of the image to draw.
sourceRect	The source rectangle of the image.
destinationRect	The destination rectangle of the image.
tag	A tag to access the SKRenderAction. If this is null this SKRenderAction is not visible in the hit test.

Returns

A new SKRenderAction representing an image.

Definition at line 397 of file SKRenderContext.cs.

7.166.3.5 InvalidateAll()

```
void VectSharp.Canvas.SKRenderAction.InvalidateAll ( )
```

This methods signals to the Parent that the Z-index, shape and visual properties (e.g. the colour) of this object have changed and triggers a redraw.

If you make changes to more than one SKRenderAction contained in the same Canvas, you only need to invalidate the last one.

This method should only be called after the output has been fully initialized.

Definition at line 303 of file SKRenderContext.cs.

7.166.3.6 InvalidateHitTestPath()

```
void VectSharp.Canvas.SKRenderAction.InvalidateHitTestPath ( )
```

Signals to this object that its shape has changed and a new path needs to be computed for the purpose of hit-testing. Also signals to the Parent that the visual properties of this object have changed and triggers a redraw.

This method should be called whenever the "shape" of the object represented by the SKRenderAction changes. If only the visual properties of this object have changed (e.g. the colour), call the InvalidateVisual method instead.

If you make changes to more than one SKRenderAction contained in the same SKMultiLayerRenderCanvas, you only need to invalidate the last one.

This method should only be called after the output has been fully initialized.

Definition at line 232 of file SKRenderContext.cs.

7.166.3.7 InvalidateVisual()

```
void VectSharp.Canvas.SKRenderAction.InvalidateVisual ( )
```

This methods signals to the Parent that the visual properties (e.g. the colour) of this object have changed and triggers a redraw.

If the "shape" of the object has changed as well, call the InvalidateHitTestPath method instead. If the Z-index of the object has changed, call the InvalidateZIndex method instead. If both the "shape" and the Z-index of the object have changed, call the InvalidateAll method.

If you make changes to more than one SKRenderAction contained in the same Canvas, you only need to invalidate the last one.

This method should only be called after the output has been fully initialized.

Definition at line 282 of file SKRenderContext.cs.

7.166.3.8 InvalidateZIndex()

```
void VectSharp.Canvas.SKRenderAction.InvalidateZIndex ( )
```

This methods signals to the Parent that the Z-index and visual properties (e.g. the colour) of this object have changed and triggers a redraw.

If the "shape" of the object has changed as well, call the InvalidateAll method instead.

If you make changes to more than one SKRenderAction contained in the same Canvas, you only need to invalidate the last one.

This method should only be called after the output has been fully initialized.

Definition at line 293 of file SKRenderContext.cs.

7.166.3.9 PathAction()

```
static SKRenderAction VectSharp.Canvas.SKRenderAction.PathAction (
            SKPath path,
            SKPaint paint,
            string tag = null )  [static]
```

Creates a new SKRenderAction representing a path.

Parameters

path	The geometry to be rendered.
paint	The paint used to fill or stroke the path.
tag	A tag to access the SKRenderAction. If this is null this SKRenderAction is not visible in the hit test.

Returns

A new SKRenderAction representing a path.

Definition at line 321 of file SKRenderContext.cs.

7.166.3.10 RestoreAction()

```
static SKRenderAction VectSharp.Canvas.SKRenderAction.RestoreAction (
            string tag = null )  [static]
```

Creates a new SKRenderAction that saves the current graphics state.

Parameters

tag	A tag to access the SKRenderAction.

Returns

A new SKRenderAction that restores the last saved graphics state.

Definition at line 451 of file SKRenderContext.cs.

7.166.3.11 SaveAction()

```
static SKRenderAction VectSharp.Canvas.SKRenderAction.SaveAction (
            string tag = null )  [static]
```

Creates a new SKRenderAction that saves the current graphics state.

Parameters

tag	A tag to access the SKRenderAction.

Returns

A new SKRenderAction that saves the current graphics state.

Definition at line 437 of file SKRenderContext.cs.

7.166.3.12 TextAction()

```
static SKRenderAction VectSharp.Canvas.SKRenderAction.TextAction (
            string text,
```

```
            float x,
            float y,
            SKFont font,
            SKPaint paint,
            string tag = null )  [static]
```

Creates a new SKRenderAction representing text.

Parameters

text	The text to be rendered.
x	The X coordinate at which the text will be drawn.
y	The Y coordinate at which the text will be drawn.
font	The font to be used to render the text.
paint	The paint to be used to fill or stroke the text.
tag	A tag to access the SKRenderAction. If this is null this SKRenderAction is not visible in the hit test.

Returns

A new SKRenderAction representing text.

Definition at line 365 of file SKRenderContext.cs.

7.166.3.13 TransformAction()

```
static SKRenderAction VectSharp.Canvas.SKRenderAction.TransformAction (
            SKMatrix transform,
            string tag = null )  [static]
```

Creates a new SKRenderAction representing a transform.

Parameters

transform	The transform to apply.
tag	A tag to access the SKRenderAction.

Returns

A new SKRenderAction representing a transform.

Definition at line 422 of file SKRenderContext.cs.

7.166.4 Property Documentation

7.166.4.1 ActionType

```
ActionTypes VectSharp.Canvas.SKRenderAction.ActionType [get]
```

Type of the rendering action.

Definition at line 88 of file SKRenderContext.cs.

7.166.4.2 Disposed

```
bool VectSharp.Canvas.SKRenderAction.Disposed [get]
```

Returns a boolean value indicating whether the current instance has been disposed.

Definition at line 35 of file SKRenderContext.cs.

7.166.4.3 Filter

```
IFilter VectSharp.Canvas.SKRenderAction.Filter [get], [set]
```

The filter with which the Graphics is drawn. If you change this, you probably want to call this object's InvalidateVisual method.

Definition at line 163 of file SKRenderContext.cs.

7.166.4.4 Font

```
SKFont VectSharp.Canvas.SKRenderAction.Font [get], [set]
```

The font that will be used to render the text (null if the action type is not ActionTypes.Text). If you change this, you probably want to call this object's InvalidateHitTestPath method.

Definition at line 106 of file SKRenderContext.cs.

7.166.4.5 Graphics

```
SKRenderContext VectSharp.Canvas.SKRenderAction.Graphics = null [get], [set]
```

The graphics that will be drawn with the specified Filter. If you change this, you probably want to call this object's InvalidateVisual method.

Definition at line 158 of file SKRenderContext.cs.

7.166.4.6 ImageDestination

```
SKRect?  VectSharp.Canvas.SKRenderAction.ImageDestination  [get], [set]
```

The destination rectangle of the image (`null` if the action type is not ActionTypes.RasterImage). If you change this, you probably want to call this object's InvalidateHitTestPath method.

Definition at line 136 of file SKRenderContext.cs.

7.166.4.7 ImageId

```
string VectSharp.Canvas.SKRenderAction.ImageId  [get], [set]
```

Univocal identifier of the image that needs to be drawn.

Definition at line 126 of file SKRenderContext.cs.

7.166.4.8 ImageSource

```
SKRect?  VectSharp.Canvas.SKRenderAction.ImageSource  [get], [set]
```

The source rectangle of the image (`null` if the action type is not ActionTypes.RasterImage). If you change this, you probably want to call this object's InvalidateVisual method.

Definition at line 131 of file SKRenderContext.cs.

7.166.4.9 Paint

```
SKPaint VectSharp.Canvas.SKRenderAction.Paint  [get], [set]
```

Paint used to render the text or path (`null` if the action type is neither ActionTypes.Text nor ActionTypes.Path). If you change this, you probably want to call this object's InvalidateHitTestPath method.

Definition at line 121 of file SKRenderContext.cs.

7.166.4.10 Parent

```
Avalonia.Controls.Canvas VectSharp.Canvas.SKRenderAction.Parent  [get]
```

The container of this SKRenderAction.

Definition at line 175 of file SKRenderContext.cs.

7.166.4.11 Path

```
SKPath VectSharp.Canvas.SKRenderAction.Path  [get], [set]
```

Path that needs to be rendered (null if the action type is not ActionTypes.Path). If you change this, you probably want to call this object's InvalidateHitTestPath method.

Definition at line 93 of file SKRenderContext.cs.

7.166.4.12 Payload

```
object VectSharp.Canvas.SKRenderAction.Payload  [get], [set]
```

An arbitrary object associated with the RenderAction.

Definition at line 168 of file SKRenderContext.cs.

7.166.4.13 Tag

```
string VectSharp.Canvas.SKRenderAction.Tag  [get]
```

A tag to access the SKRenderAction.

Definition at line 146 of file SKRenderContext.cs.

7.166.4.14 Text

```
string VectSharp.Canvas.SKRenderAction.Text  [get], [set]
```

Text that needs to be rendered (null if the action type is not ActionTypes.Text). If you change this, you probably want to call this object's InvalidateHitTestPath method.

Definition at line 101 of file SKRenderContext.cs.

7.166.4.15 TextX

```
float VectSharp.Canvas.SKRenderAction.TextX  [get], [set]
```

The X coordainate at which the text will be drawn (null if the action type is not ActionTypes.Text). If you change this, you probably want to call this object's InvalidateHitTestPath method.

Definition at line 111 of file SKRenderContext.cs.

7.166.4.16 TextY

```
float VectSharp.Canvas.SKRenderAction.TextY  [get], [set]
```

The Y coordainate at which the text will be drawn (null if the action type is not ActionTypes.Text). If you change this, you probably want to call this object's InvalidateHitTestPath method.

Definition at line 116 of file SKRenderContext.cs.

7.166.4.17 Transform

```
SKMatrix?  VectSharp.Canvas.SKRenderAction.Transform = null  [get], [set]
```

The transformation matrix that will be applied to the current coordinate system (`null` if the action type is not ActionTypes.Transform). If you change this, you probably want to call this object's InvalidateVisual method.

Definition at line 141 of file SKRenderContext.cs.

7.166.4.18 ZIndex

```
uint VectSharp.Canvas.SKRenderAction.ZIndex = 0  [get], [set]
```

The Z-index of the rendering action (an action with a higher Z-index will always appear above an action with a lower Z-index). The more different values there are for the Z-index, the slower the rendering, so keep use of this property to a minimum. If you change this, you probably want to call this object's InvalidateZIndex method.

Definition at line 153 of file SKRenderContext.cs.

7.166.5 Event Documentation

7.166.5.1 PointerEntered

```
EventHandler<Avalonia.Input.PointerEventArgs> VectSharp.Canvas.SKRenderAction.PointerEntered
```

Raised when the pointer enters the area covered by the SKRenderAction.

Definition at line 186 of file SKRenderContext.cs.

7.166.5.2 PointerExited

```
EventHandler<Avalonia.Input.PointerEventArgs> VectSharp.Canvas.SKRenderAction.PointerExited
```

Raised when the pointer leaves the area covered by the SKRenderAction.

Definition at line 191 of file SKRenderContext.cs.

7.166.5.3 PointerPressed

```
EventHandler<Avalonia.Input.PointerPressedEventArgs> VectSharp.Canvas.SKRenderAction.Pointer↩
Pressed
```

Raised when the pointer is pressed while over the area covered by the SKRenderAction.

Definition at line 196 of file SKRenderContext.cs.

7.166.5.4 PointerReleased

```
EventHandler<Avalonia.Input.PointerReleasedEventArgs> VectSharp.Canvas.SKRenderAction.Pointer↩
Released
```

Raised when the pointer is released after a PointerPressed event.

Definition at line 201 of file SKRenderContext.cs.

The documentation for this class was generated from the following file:

- VectSharp.Canvas/SKRenderContext.cs

7.167 VectSharp.Canvas.SKRenderContext Class Reference

Represents a page that has been prepared for fast rendering using the SkiaSharp renderer.

7.167.1 Detailed Description

Represents a page that has been prepared for fast rendering using the SkiaSharp renderer.

Definition at line 546 of file SKRenderContext.cs.

The documentation for this class was generated from the following file:

- VectSharp.Canvas/SKRenderContext.cs

7.168 VectSharp.Canvas.SKRenderContextInterpreter Class Reference

Contains methods to render a Page to an Avalonia.Controls.Canvas using the SkiaSharp renderer.

Static Public Member Functions

- static SKMultiLayerRenderCanvas PaintToSKCanvas (this Document document, double? width=null, double? height=null, Colour? background=null, AvaloniaContextInterpreter.TextOptions textOption=AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption=default)

  Render a Document to an Avalonia.Controls.Canvas using the SkiaSharp renderer. Each page corresponds to a layer in the image.
- static SKMultiLayerRenderCanvas PaintToSKCanvas (this Document document, Dictionary< string, Func< SKRenderAction, IEnumerable< SKRenderAction > > > taggedActions, bool removeTaggedActionsAfterExecution=true, double? width=null, double? height=null, Colour? background=null, AvaloniaContextInterpreter.TextOptions textOption=AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption=default)

  Render a Document to an Avalonia.Controls.Canvas using the SkiaSharp renderer. Each page corresponds to a layer in the image.
- static SKMultiLayerRenderCanvas PaintToSKCanvas (this Document document, Dictionary< string, Func< SKRenderAction, IEnumerable< SKRenderAction > > > taggedActions, Dictionary< string,(SKBitmap, bool)> images, bool removeTaggedActionsAfterExecution=true, double? width=null, double? height=null, Colour? background=null, AvaloniaContextInterpreter.TextOptions textOption=AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption=default)

  Render a Document to an Avalonia.Controls.Canvas using the SkiaSharp renderer. Each page corresponds to a layer in the image.
- static SKMultiLayerRenderCanvas PaintToSKCanvas (this Page page, AvaloniaContextInterpreter.TextOptions textOption=AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption=default)

  Render a Page to an Avalonia.Controls.Canvas using the SkiaSharp renderer.
- static SKMultiLayerRenderCanvas PaintToSKCanvas (this Page page, Dictionary< string, Func< SKRenderAction, IEnumerable< SKRenderAction > > > taggedActions, bool removeTaggedActionsAfterExecution=true, AvaloniaContextInterpreter.TextOptions textOption=AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption=default)

  Render a Page to an Avalonia.Controls.Canvas using the SkiaSharpRenderer.
- static SKMultiLayerRenderCanvas PaintToSKCanvas (this Page page, Dictionary< string, Func< SKRenderAction, IEnumerable< SKRenderAction > > > taggedActions, Dictionary< string,(SKBitmap, bool)> images, bool removeTaggedActionsAfterExecution=true, AvaloniaContextInterpreter.TextOptions textOption=AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption=default)

  Render a Page to an Avalonia.Controls.Canvas using the SkiaSharpRenderer.
- static SKRenderContext CopyToSKRenderContext (this Page page, AvaloniaContextInterpreter.TextOptions textOption=AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption=default)

  Render a Page to a SKRenderContext. This can be drawn using the SkiaSharpRenderer by adding it to a SKMultiLayerRenderCanvas.
- static SKRenderContext CopyToSKRenderContext (this Page page, Dictionary< string, Func< SKRenderAction, IEnumerable< SKRenderAction > > > taggedActions, bool removeTaggedActionsAfterExecution=true, AvaloniaContextInterpreter.TextOptions textOption=AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption=default)

  Render a Page to a SKRenderContext. This can be drawn using the SkiaSharpRenderer by adding it to a SKMultiLayerRenderCanvas.
- static SKRenderContext CopyToSKRenderContext (this Page page, Dictionary< string, Func< SKRenderAction, IEnumerable< SKRenderAction > > > taggedActions, Dictionary< string,(SKBitmap, bool)> images, bool removeTaggedActionsAfterExecution=true, AvaloniaContextInterpreter.TextOptions textOption=AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption=default)

  Render a Page to a SKRenderContext. This can be drawn using the SkiaSharpRenderer by adding it to a SKMultiLayerRenderCanvas.
- static RasterImage Rasterise (this Graphics graphics, Rectangle region, double scale, bool interpolate)

  Rasterise a region of a Graphics object.

7.168.1 Detailed Description

Contains methods to render a Page to an Avalonia.Controls.Canvas using the SkiaSharp renderer.

Definition at line 1545 of file SKRenderContext.cs.

7.168.2 Member Function Documentation

7.168.2.1 CopyToSKRenderContext() [1/3]

```
static SKRenderContext VectSharp.Canvas.SKRenderContextInterpreter.CopyToSKRenderContext (
            this Page page,
            AvaloniaContextInterpreter.TextOptions textOption = AvaloniaContextInterpreter.TextOptions.Conver
            FilterOption filterOption = default )  [static]
```

Render a Page to a SKRenderContext. This can be drawn using the SkiaSharpRenderer by adding it to a SKMultiLayerRenderCanvas.

Parameters

page	The Page to render.
textOption	Defines whether text items should be converted into paths when drawing.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

A SKRenderContext containing the rendered graphics objects.

Definition at line 1678 of file SKRenderContext.cs.

7.168.2.2 CopyToSKRenderContext() [2/3]

```
static SKRenderContext VectSharp.Canvas.SKRenderContextInterpreter.CopyToSKRenderContext (
            this Page page,
            Dictionary< string, Func< SKRenderAction, IEnumerable< SKRenderAction > > >
taggedActions,
            bool removeTaggedActionsAfterExecution = true,
            AvaloniaContextInterpreter.TextOptions textOption = AvaloniaContextInterpreter.TextOptions.Conver
            FilterOption filterOption = default )  [static]
```

Render a Page to a SKRenderContext. This can be drawn using the SkiaSharpRenderer by adding it to a SKMultiLayerRenderCanvas.

Parameters

page	The Page to render.
taggedActions	A Dictionary containing the actions that will be performed on items with the corresponding tag. These should be functions that accept one parameter of type SKRenderAction and return an IEnumerable<SKRenderAction> of the render actions that will actually be added to the plot.
removeTaggedActionsAfterExecution	Whether the actions should be removed from *taggedActions* after their execution. Set to false if the same action should be performed on multiple items with the same tag.
textOption	Defines whether text items should be converted into paths when drawing.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

A SKRenderContext containing the rendered graphics objects.

Definition at line 1695 of file SKRenderContext.cs.

7.168.2.3 CopyToSKRenderContext() [3/3]

```
static SKRenderContext VectSharp.Canvas.SKRenderContextInterpreter.CopyToSKRenderContext (
            this Page page,
            Dictionary< string, Func< SKRenderAction, IEnumerable< SKRenderAction > > >
taggedActions,
            Dictionary< string,(SKBitmap, bool)> images,
            bool removeTaggedActionsAfterExecution = true,
            AvaloniaContextInterpreter.TextOptions textOption = AvaloniaContextInterpreter.TextOptions.Conver
            FilterOption filterOption = default )  [static]
```

Render a Page to a SKRenderContext. This can be drawn using the SkiaSharpRenderer by adding it to a SKMultiLayerRenderCanvas.

Parameters

page	The Page to render.
taggedActions	A Dictionary containing the actions that will be performed on items with the corresponding tag. These should be functions that accept one parameter of type SKRenderAction and return an IEnumerable<SKRenderAction> of the render actions that will actually be added to the plot.
images	A dictionary that associates to each raster image path (or data URL) the image rendered as a SKBitmap and a boolean value indicating whether it should be drawn as "pixelated" or not. This will be populated automatically as the page is rendered. If you are rendering multiple Pages (or you are rendering the same page multiple times), it will be beneficial to keep a reference to this dictionary and pass it again on further rendering requests; otherwise, you can just pass an empty dictionary.

Parameters

removeTaggedActionsAfterExecution	Whether the actions should be removed from *taggedActions* after their execution. Set to false if the same action should be performed on multiple items with the same tag.
textOption	Defines whether text items should be converted into paths when drawing.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

A SKRenderContext containing the rendered graphics objects.

Definition at line 1714 of file SKRenderContext.cs.

7.168.2.4 PaintToSKCanvas() [1/6]

```
static SKMultiLayerRenderCanvas VectSharp.Canvas.SKRenderContextInterpreter.PaintToSKCanvas (
            this Document document,
            Dictionary< string, Func< SKRenderAction, IEnumerable< SKRenderAction > > >
taggedActions,
            bool removeTaggedActionsAfterExecution = true,
            double?  width = null,
            double?  height = null,
            Colour?  background = null,
            AvaloniaContextInterpreter.TextOptions textOption = AvaloniaContextInterpreter.TextOptions.Conver
            FilterOption filterOption = default )  [static]
```

Render a Document to an Avalonia.Controls.Canvas using the SkiaSharp renderer. Each page corresponds to a layer in the image.

Parameters

document	The Document to render.
taggedActions	A Dictionary containing the actions that will be performed on items with the corresponding tag. These should be functions that accept one parameter of type SKRenderAction and return an IEnumerable<SKRenderAction> of the render actions that will actually be added to the plot.
removeTaggedActionsAfterExecution	Whether the actions should be removed from *taggedActions* after their execution. Set to false if the same action should be performed on multiple items with the same tag.
width	The width of the document. If this is `null`, the width of the largest page is used.
height	The height of the document. If this is `null`, the height of the largest page is used.
background	The background colour of the document. If this is `null`, a transparent background is used.
textOption	Defines whether text items should be converted into paths when drawing.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

An Avalonia.Controls.Canvas containing the rendered graphics objects.

Definition at line 1592 of file SKRenderContext.cs.

7.168.2.5 PaintToSKCanvas() [2/6]

```
static SKMultiLayerRenderCanvas VectSharp.Canvas.SKRenderContextInterpreter.PaintToSKCanvas (
            this Document document,
            Dictionary< string, Func< SKRenderAction, IEnumerable< SKRenderAction > > >
taggedActions,
            Dictionary< string,(SKBitmap, bool)> images,
            bool removeTaggedActionsAfterExecution = true,
            double?  width = null,
            double?  height = null,
            Colour?  background = null,
            AvaloniaContextInterpreter.TextOptions textOption = AvaloniaContextInterpreter.TextOptions.Conver
            FilterOption filterOption = default )  [static]
```

Render a Document to an Avalonia.Controls.Canvas using the SkiaSharp renderer. Each page corresponds to a layer in the image.

Parameters

document	The Document to render.
taggedActions	A Dictionary containing the actions that will be performed on items with the corresponding tag. These should be functions that accept one parameter of type SKRenderAction and return an IEnumerable<SKRenderAction> of the render actions that will actually be added to the plot.
images	A dictionary that associates to each raster image path (or data URL) the image rendered as a SKBitmap and a boolean value indicating whether it should be drawn as "pixelated" or not. This will be populated automatically as the page is rendered. If you are rendering multiple Pages (or you are rendering the same page multiple times), it will be beneficial to keep a reference to this dictionary and pass it again on further rendering requests; otherwise, you can just pass an empty dictionary.
removeTaggedActionsAfterExecution	Whether the actions should be removed from *taggedActions* after their execution. Set to false if the same action should be performed on multiple items with the same tag.
width	The width of the document. If this is `null`, the width of the largest page is used.
height	The height of the document. If this is `null`, the height of the largest page is used.
background	The background colour of the document. If this is `null`, a transparent background is used.
textOption	Defines whether text items should be converted into paths when drawing.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

An Avalonia.Controls.Canvas containing the rendered graphics objects.

Definition at line 1614 of file SKRenderContext.cs.

7.168.2.6 PaintToSKCanvas() [3/6]

```
static SKMultiLayerRenderCanvas VectSharp.Canvas.SKRenderContextInterpreter.PaintToSKCanvas (
            this Document document,
            double?  width = null,
            double?  height = null,
            Colour?  background = null,
            AvaloniaContextInterpreter.TextOptions textOption = AvaloniaContextInterpreter.TextOptions.Conver
            FilterOption filterOption = default )  [static]
```

Render a Document to an Avalonia.Controls.Canvas using the SkiaSharp renderer. Each page corresponds to a layer in the image.

Parameters

document	The Document to render.
width	The width of the document. If this is `null`, the width of the largest page is used.
height	The height of the document. If this is `null`, the height of the largest page is used.
background	The background colour of the document. If this is `null`, a transparent background is used.
textOption	Defines whether text items should be converted into paths when drawing.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

An Avalonia.Controls.Canvas containing the rendered graphics objects.

Definition at line 1572 of file SKRenderContext.cs.

7.168.2.7 PaintToSKCanvas() [4/6]

```
static SKMultiLayerRenderCanvas VectSharp.Canvas.SKRenderContextInterpreter.PaintToSKCanvas (
            this Page page,
            AvaloniaContextInterpreter.TextOptions textOption = AvaloniaContextInterpreter.TextOptions.Conver
            FilterOption filterOption = default )  [static]
```

Render a Page to an Avalonia.Controls.Canvas using the SkiaSharp renderer.

Parameters

page	The Page to render.
textOption	Defines whether text items should be converted into paths when drawing.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

An Avalonia.Controls.Canvas containing the rendered graphics objects.

Definition at line 1628 of file SKRenderContext.cs.

7.168.2.8 PaintToSKCanvas() [5/6]

```
static SKMultiLayerRenderCanvas VectSharp.Canvas.SKRenderContextInterpreter.PaintToSKCanvas (
            this Page page,
            Dictionary< string, Func< SKRenderAction, IEnumerable< SKRenderAction > > >
taggedActions,
            bool removeTaggedActionsAfterExecution = true,
            AvaloniaContextInterpreter.TextOptions textOption = AvaloniaContextInterpreter.TextOptions.Conver
            FilterOption filterOption = default )  [static]
```

Render a Page to an Avalonia.Controls.Canvas using the SkiaSharpRenderer.

Parameters

page	The Page to render.
taggedActions	A Dictionary containing the actions that will be performed on items with the corresponding tag. These should be functions that accept one parameter of type SKRenderAction and return an IEnumerable<SKRenderAction> of the render actions that will actually be added to the plot.
removeTaggedActionsAfterExecution	Whether the actions should be removed from *taggedActions* after their execution. Set to false if the same action should be performed on multiple items with the same tag.
textOption	Defines whether text items should be converted into paths when drawing.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

An Avalonia.Controls.Canvas containing the rendered graphics objects.

Definition at line 1645 of file SKRenderContext.cs.

7.168.2.9 PaintToSKCanvas() [6/6]

```
static SKMultiLayerRenderCanvas VectSharp.Canvas.SKRenderContextInterpreter.PaintToSKCanvas (
            this Page page,
            Dictionary< string, Func< SKRenderAction, IEnumerable< SKRenderAction > > >
taggedActions,
            Dictionary< string,(SKBitmap, bool)> images,
```

```
bool removeTaggedActionsAfterExecution = true,
AvaloniaContextInterpreter.TextOptions textOption = AvaloniaContextInterpreter.TextOptions.Conver
FilterOption filterOption = default )  [static]
```

Render a Page to an Avalonia.Controls.Canvas using the SkiaSharpRenderer.

Parameters

page	The Page to render.
taggedActions	A Dictionary containing the actions that will be performed on items with the corresponding tag. These should be functions that accept one parameter of type SKRenderAction and return an IEnumerable<SKRenderAction> of the render actions that will actually be added to the plot.
images	A dictionary that associates to each raster image path (or data URL) the image rendered as a SKBitmap and a boolean value indicating whether it should be drawn as "pixelated" or not. This will be populated automatically as the page is rendered. If you are rendering multiple Pages (or you are rendering the same page multiple times), it will be beneficial to keep a reference to this dictionary and pass it again on further rendering requests; otherwise, you can just pass an empty dictionary.
removeTaggedActionsAfterExecution	Whether the actions should be removed from *taggedActions* after their execution. Set to false if the same action should be performed on multiple items with the same tag.
textOption	Defines whether text items should be converted into paths when drawing.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

An Avalonia.Controls.Canvas containing the rendered graphics objects.

Definition at line 1664 of file SKRenderContext.cs.

7.168.2.10 Rasterise()

```
static RasterImage VectSharp.Canvas.SKRenderContextInterpreter.Rasterise (
            this Graphics graphics,
            Rectangle region,
            double scale,
            bool interpolate )  [static]
```

Rasterise a region of a Graphics object.

Parameters

graphics	The Graphics object that will be rasterised.
region	The region of the *graphics* that will be rasterised.
scale	The scale at which the image will be rendered.
interpolate	Whether the resulting image should be interpolated or not when it is drawn on another Graphics surface.

Returns

A RasterImage containing the rasterised graphics.

Definition at line 1736 of file SKRenderContext.cs.

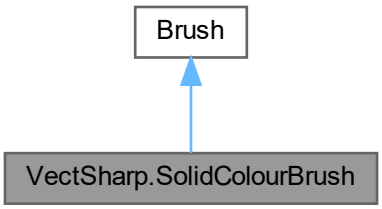
The documentation for this class was generated from the following file:

- VectSharp.Canvas/SKRenderContext.cs

7.169 VectSharp.SolidColourBrush Class Reference

Represents a brush painting with a single solid colour.

Inheritance diagram for VectSharp.SolidColourBrush:



Public Member Functions

- SolidColourBrush (Colour colour)

  Creates a new SolidColourBrush with the specified colour .
- override Brush MultiplyOpacity (double opacity)

  Returns a brush corresponding the current instance, with the specified opacity multiplication applied.

Parameters

opacity	*The value that will be used to multiply the opacity of the brush.*

Returns

A brush corresponding the current instance, with the specified opacity multiplication applied.

Static Public Member Functions

- static implicit operator SolidColourBrush (Colour colour)

  Implicitly converts a Colour into a SolidColourBrush.

Properties

- Colour Colour [get]

  The colour of the brush.
- double R [get]

  Red component of the colour. Range: [0, 1].
- double G [get]

  Green component of the colour. Range: [0, 1].
- double B [get]

  Blue component of the colour. Range: [0, 1].
- double A [get]

  Alpha component of the colour. Range: [0, 1].

7.169.1 Detailed Description

Represents a brush painting with a single solid colour.

Definition at line 57 of file Brush.cs.

7.169.2 Constructor & Destructor Documentation

7.169.2.1 SolidColourBrush()

```
VectSharp.SolidColourBrush.SolidColourBrush (
            Colour colour )
```

Creates a new SolidColourBrush with the specified *colour* .

Parameters

colour	The Colour to use for the brush.

Definition at line 88 of file Brush.cs.

7.169.3 Member Function Documentation

7.169.3.1 MultiplyOpacity()

```
override Brush VectSharp.SolidColourBrush.MultiplyOpacity (
            double opacity )  [virtual]
```

Returns a brush corresponding the current instance, with the specified *opacity* multiplication applied.

Parameters

opacity	The value that will be used to multiply the opacity of the brush.

Returns

A brush corresponding the current instance, with the specified *opacity* multiplication applied.

Implements VectSharp.Brush.

Definition at line 94 of file Brush.cs.

7.169.3.2 operator SolidColourBrush()

```
static implicit VectSharp.SolidColourBrush.operator SolidColourBrush (
            Colour colour )  [static]
```

Implicitly converts a Colour into a SolidColourBrush.

Parameters

colour	The Colour to use for the brush.

Definition at line 103 of file Brush.cs.

7.169.4 Property Documentation

7.169.4.1 A

```
double VectSharp.SolidColourBrush.A  [get]
```

Alpha component of the colour. Range: [0, 1].

Definition at line 82 of file Brush.cs.

7.169.4.2 B

```
double VectSharp.SolidColourBrush.B  [get]
```

Blue component of the colour. Range: [0, 1].

Definition at line 77 of file Brush.cs.

7.169.4.3 Colour

```
Colour VectSharp.SolidColourBrush.Colour  [get]
```

The colour of the brush.

Definition at line 62 of file Brush.cs.

7.169.4.4 G

```
double VectSharp.SolidColourBrush.G  [get]
```

Green component of the colour. Range: [0, 1].

Definition at line 72 of file Brush.cs.

7.169.4.5 R

```
double VectSharp.SolidColourBrush.R  [get]
```

Red component of the colour. Range: [0, 1].

Definition at line 67 of file Brush.cs.

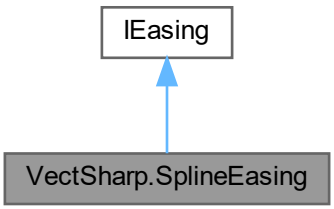
The documentation for this class was generated from the following file:

- VectSharp/Brush.cs

7.170 VectSharp.SplineEasing Class Reference

Describes an easing defined by a Cubic Bezier curve.

Inheritance diagram for VectSharp.SplineEasing:



Public Member Functions

- SplineEasing (Point controlPoint1, Point controlPoint2)

  Creates a new SplineEasing with the specified control points. The start point is always (0, 0) and the end point is always (1, 1).
- double Ease (double value)

  Applies the easing to the specified transition offset.

Parameters

value	*The transition offset (ranging from 0 to 1).*

Returns

The eased transition offset value.

Properties

- Point ControlPoint1 `[get]`

  The first control point of the curve.
- Point ControlPoint2 `[get]`

  The second control point of the curve.

7.170.1 Detailed Description

Describes an easing defined by a Cubic Bezier curve.

Definition at line 1261 of file Animation.cs.

7.170.2 Constructor & Destructor Documentation

7.170.2.1 SplineEasing()

```
VectSharp.SplineEasing.SplineEasing (
            Point controlPoint1,
            Point controlPoint2 )
```

Creates a new SplineEasing with the specified control points. The start point is always (0, 0) and the end point is always (1, 1).

Parameters

controlPoint1	The first control point of the curve. Both X and Y must be between 0 and 1, inclusive.
controlPoint2	The second control point of the curve. Both X and Y must be between 0 and 1, inclusive.

Exceptions

ArgumentException	This exception is thrown if any coordinate of the control points is < 0 or > 1.

Definition at line 1280 of file Animation.cs.

7.170.3 Member Function Documentation

7.170.3.1 Ease()

```
double VectSharp.SplineEasing.Ease (
            double value )
```

Applies the easing to the specified transition offset.

Parameters

value	The transition offset (ranging from 0 to 1).

Returns

The eased transition offset value.

Implements VectSharp.IEasing.

Definition at line 1330 of file Animation.cs.

7.170.4 Property Documentation

7.170.4.1 ControlPoint1

```
Point VectSharp.SplineEasing.ControlPoint1  [get]
```

The first control point of the curve.

Definition at line 1266 of file Animation.cs.

7.170.4.2 ControlPoint2

```
Point VectSharp.SplineEasing.ControlPoint2  [get]
```

The second control point of the curve.

Definition at line 1271 of file Animation.cs.

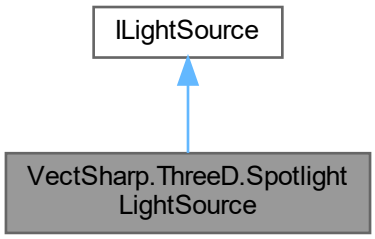
The documentation for this class was generated from the following file:

- VectSharp/Animation.cs

7.171 VectSharp.ThreeD.SpotlightLightSource Class Reference

Represents a conic spotlight.

Inheritance diagram for VectSharp.ThreeD.SpotlightLightSource:



Public Member Functions

- SpotlightLightSource (double intensity, Point3D position, NormalizedVector3D direction, double beamWidth↩ Angle, double cutoffAngle)

  Creates a new SpotlightLightSource instance.
- LightIntensity GetLightAt (Point3D point)

  Computes the light intensity at the specified point, without taking into account any obstructions.

Parameters

point	*The Point3DElement at which the light intensity should be computed.*

Returns

- double GetObstruction (Point3D point, IEnumerable< Triangle3DElement > shadowingTriangles)

  Determines the amount of obstruction of the light that results at point due to the specified shadowingTriangles .

Parameters

point	*The Point3D at which the obstruction should be computed.*
shadowingTriangles	*A collection of Triangle3DElement casting shadows.*

Returns

1 if the light is completely obstructed, 0 if the light is completely visible, a value between these if the light is partially obstructed.

Properties

- bool CastsShadow = true `[get, set]`

  Determines whether the light casts a shadow or not.
- Point3D Position `[get, set]`

The position of the light source.

- NormalizedVector3D Direction `[get, set]`

  The direction of the cone axis.

- double Intensity `[get, set]`

  The base intensity of the light.

- double BeamWidthAngle `[get, set]`

  The angular size of the light cone, in radians.

- double CutoffAngle `[get, set]`

  The angular size of the cutoff cone, in radians.

- double DistanceAttenuationExponent = 2 `[get, set]`

  An exponent determining how fast the light attenuates with increasing distance. Set to 0 to disable distance attenuation.

- double AngleAttenuationExponent = 1 `[get, set]`

  An exponent determining how fast the light attenuates between the main light cone and the cutoff cone.

7.171.1 Detailed Description

Represents a conic spotlight.

Definition at line 256 of file Lights.cs.

7.171.2 Constructor & Destructor Documentation

7.171.2.1 SpotlightLightSource()

```
VectSharp.ThreeD.SpotlightLightSource.SpotlightLightSource (
            double intensity,
            Point3D position,
            NormalizedVector3D direction,
            double beamWidthAngle,
            double cutoffAngle )
```

Creates a new SpotlightLightSource instance.

Parameters

intensity	The intensity of the light.
position	The position of the light source.
direction	The direction of the cone's axis.
beamWidthAngle	The angular size of the light cone, in radians.
cutoffAngle	The angular size of the cutoff cone, in radians.

Definition at line 304 of file Lights.cs.

7.171.3 Member Function Documentation

7.171.3.1 GetLightAt()

```
LightIntensity VectSharp.ThreeD.SpotlightLightSource.GetLightAt (
            Point3D point )
```

Computes the light intensity at the specified point, without taking into account any obstructions.

Parameters

point	The Point3DElement at which the light intensity should be computed.

Returns

Implements VectSharp.ThreeD.ILightSource.

Definition at line 314 of file Lights.cs.

7.171.3.2 GetObstruction()

```
double VectSharp.ThreeD.SpotlightLightSource.GetObstruction (
            Point3D point,
            IEnumerable< Triangle3DElement > shadowingTriangles )
```

Determines the amount of obstruction of the light that results at *point* due to the specified *shadowingTriangles* .

Parameters

point	The Point3D at which the obstruction should be computed.
shadowingTriangles	A collection of Triangle3DElement casting shadows.

Returns

1 if the light is completely obstructed, 0 if the light is completely visible, a value between these if the light is partially obstructed.

Implements VectSharp.ThreeD.ILightSource.

Definition at line 362 of file Lights.cs.

7.171.4 Property Documentation

7.171.4.1 AngleAttenuationExponent

```
double VectSharp.ThreeD.SpotlightLightSource.AngleAttenuationExponent = 1  [get], [set]
```

An exponent determining how fast the light attenuates between the main light cone and the cutoff cone.

Definition at line 294 of file Lights.cs.

7.171.4.2 BeamWidthAngle

```
double VectSharp.ThreeD.SpotlightLightSource.BeamWidthAngle  [get], [set]
```

The angular size of the light cone, in radians.

Definition at line 279 of file Lights.cs.

7.171.4.3 CastsShadow

```
bool VectSharp.ThreeD.SpotlightLightSource.CastsShadow = true  [get], [set]
```

Determines whether the light casts a shadow or not.

Implements VectSharp.ThreeD.ILightSource.

Definition at line 259 of file Lights.cs.

7.171.4.4 CutoffAngle

```
double VectSharp.ThreeD.SpotlightLightSource.CutoffAngle  [get], [set]
```

The angular size of the cutoff cone, in radians.

Definition at line 284 of file Lights.cs.

7.171.4.5 Direction

```
NormalizedVector3D VectSharp.ThreeD.SpotlightLightSource.Direction  [get], [set]
```

The direction of the cone axis.

Definition at line 269 of file Lights.cs.

7.171.4.6 DistanceAttenuationExponent

```
double VectSharp.ThreeD.SpotlightLightSource.DistanceAttenuationExponent = 2  [get], [set]
```

An exponent determining how fast the light attenuates with increasing distance. Set to 0 to disable distance attenuation.

Definition at line 289 of file Lights.cs.

7.171.4.7 Intensity

```
double VectSharp.ThreeD.SpotlightLightSource.Intensity  [get], [set]
```

The base intensity of the light.

Definition at line 274 of file Lights.cs.

7.171.4.8 Position

```
Point3D VectSharp.ThreeD.SpotlightLightSource.Position  [get], [set]
```

The position of the light source.

Definition at line 264 of file Lights.cs.

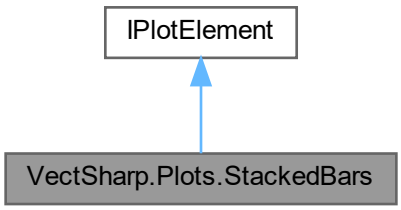
The documentation for this class was generated from the following file:

- VectSharp.ThreeD/Lights.cs

7.172 VectSharp.Plots.StackedBars Class Reference

A plot element that draws stacked bars.

Inheritance diagram for VectSharp.Plots.StackedBars:



Public Member Functions

- StackedBars (IEnumerable< IReadOnlyList< double > > data, IComparer< IReadOnlyList< double > > sorting, Func< IReadOnlyList< double >, IReadOnlyList< double > > getBaseline, ICoordinateSystem< IReadOnlyList< double > > coordinateSystem)

  Create a new StackedBars instance.
- StackedBars (IEnumerable< IReadOnlyList< double > > data, Comparison< IReadOnlyList< double > > sorting, Func< IReadOnlyList< double >, IReadOnlyList< double > > getBaseline, ICoordinateSystem< IReadOnlyList< double > > coordinateSystem)

  Create a new StackedBars instance.
- StackedBars (IEnumerable< IReadOnlyList< double > > data, ICoordinateSystem< IReadOnlyList< double > > coordinateSystem, bool vertical=true)

  Create a new StackedBars instance.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- bool Vertical = true `[get, set]`

  If this is `true`, the bars rise vertically above the X axis Otherwise, the bars grow horizontally from the Y axis.
- SortedSet< IReadOnlyList< double > > Data `[get, set]`

  The data points corresponding to the tips of the bars. For each bar stack, the data point contains an element determining the position of the bar on the X axis (if Vertical is `true`, or on the Y axis otherwise), and a set of elements determining the length of each segment in the bar stack.
- Func< IReadOnlyList< double >, IReadOnlyList< double > > GetBaseline `[get, set]`

  A function that returns the bottom for each bar. This function should accept a single parameter (an IReadOnlyList<T> of `double`s), and return another object of the same type, representing the bottom of the bar in data space.
- double Margin `[get, set]`

  The margin between consecutive bars. This should range between 0 and 1.
- ICoordinateSystem< IReadOnlyList< double > > CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.
- IReadOnlyList< PlotElementPresentationAttributes > PresentationAttributes = new PlotElementPresentationAttributes[ ] { new PlotElementPresentationAttributes() } `[get, set]`

  Presentation attributes for the bars. An element from this collection is used for each segment in the stack; if there are more segments than elements in this collection, the presentation attributes are wrapped.
- string Tag `[get, set]`

  A tag to identify the stacked bars in the plot.

7.172.1 Detailed Description

A plot element that draws stacked bars.

Definition at line 325 of file Bars.cs.

7.172.2 Constructor & Destructor Documentation

7.172.2.1 StackedBars() [1/3]

```
VectSharp.Plots.StackedBars.StackedBars (
            IEnumerable< IReadOnlyList< double > > data,
            IComparer< IReadOnlyList< double > > sorting,
            Func< IReadOnlyList< double >, IReadOnlyList< double > > getBaseline,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem )
```

Create a new StackedBars instance.

Parameters

data	The data points corresponding to the tips of the bars. For each bar stack, the data point contains an element determining the position of the bar on the X axis (if Vertical is `true`, or on the Y axis otherwise), and a set of elements determining the length of each segment in the bar stack.
sorting	A comparer used to sort the bars.
getBaseline	A function that returns the bottom for each bar. This function should accept a single parameter (an IReadOnlyList<T> of `double`s), and return another object of the same type, representing the bottom of the bar in data space.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 399 of file Bars.cs.

7.172.2.2 StackedBars() [2/3]

```
VectSharp.Plots.StackedBars.StackedBars (
            IEnumerable< IReadOnlyList< double > > data,
            Comparison< IReadOnlyList< double > > sorting,
            Func< IReadOnlyList< double >, IReadOnlyList< double > > getBaseline,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem )
```

Create a new StackedBars instance.

Parameters

data	The data points corresponding to the tips of the bars. For each bar stack, the data point contains an element determining the position of the bar on the X axis (if Vertical is `true`, or on the Y axis otherwise), and a set of elements determining the length of each segment in the bar stack.
sorting	A comparer used to sort the bars.
getBaseline	A function that returns the bottom for each bar. This function should accept a single parameter (an IReadOnlyList<T> of `double`s), and return another object of the same type, representing the bottom of the bar in data space.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 418 of file Bars.cs.

7.172.2.3 StackedBars() [3/3]

```
VectSharp.Plots.StackedBars.StackedBars (
            IEnumerable< IReadOnlyList< double > > data,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem,
            bool vertical = true )
```

Create a new StackedBars instance.

Parameters

data	The data points corresponding to the tips of the bars. For each bar stack, the data point contains an element determining the position of the bar on the X axis (if *vertical* is `true`, or on the Y axis otherwise), and a set of elements determining the length of each segment in the bar stack.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.
vertical	If this is `true` (the default), the bars rise vertically above the X axis Otherwise, the bars grow horizontally from the Y axis.

Definition at line 430 of file Bars.cs.

7.172.3 Member Function Documentation

7.172.3.1 Plot()

```
void VectSharp.Plots.StackedBars.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 433 of file Bars.cs.

7.172.4 Property Documentation

7.172.4.1 CoordinateSystem

```
ICoordinateSystem<IReadOnlyList<double> > VectSharp.Plots.StackedBars.CoordinateSystem  [get],
[set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 372 of file Bars.cs.

7.172.4.2 Data

```
SortedSet<IReadOnlyList<double> > VectSharp.Plots.StackedBars.Data  [get], [set]
```

The data points corresponding to the tips of the bars. For each bar stack, the data point contains an element determining the position of the bar on the X axis (if Vertical is `true`, or on the Y axis otherwise), and a set of elements determining the length of each segment in the bar stack.

Definition at line 341 of file Bars.cs.

7.172.4.3 GetBaseline

```
Func<IReadOnlyList<double>, IReadOnlyList<double> > VectSharp.Plots.StackedBars.GetBaseline
[get], [set]
```

A function that returns the bottom for each bar. This function should accept a single parameter (an IReadOnly↩ List<T> of `double`s), and return another object of the same type, representing the bottom of the bar in data space.

Definition at line 348 of file Bars.cs.

7.172.4.4 Margin

```
double VectSharp.Plots.StackedBars.Margin  [get], [set]
```

The margin between consecutive bars. This should range between 0 and 1.

Definition at line 353 of file Bars.cs.

7.172.4.5 PresentationAttributes

```
IReadOnlyList<PlotElementPresentationAttributes> VectSharp.Plots.StackedBars.Presentation↩
Attributes = new PlotElementPresentationAttributes[] { new PlotElementPresentationAttributes()
}  [get], [set]
```

Presentation attributes for the bars. An element from this collection is used for each segment in the stack; if there are more segments than elements in this collection, the presentation attributes are wrapped.

Definition at line 380 of file Bars.cs.

7.172.4.6 Tag

```
string VectSharp.Plots.StackedBars.Tag  [get], [set]
```

A tag to identify the stacked bars in the plot.

Definition at line 385 of file Bars.cs.

7.172.4.7 Vertical

```
bool VectSharp.Plots.StackedBars.Vertical = true  [get], [set]
```

If this is `true`, the bars rise vertically above the X axis Otherwise, the bars grow horizontally from the Y axis.

Definition at line 333 of file Bars.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/Bars.cs

7.173 VectSharp.SVG.SVGContextInterpreter Class Reference

Contains methods to render a Page as an SVG file.

Classes

- class FilterOption
  Determines how and whether image filters are rasterised.

Public Types

- enum TextOptions
  Defines whether the used fonts should be included in the file.

Static Public Member Functions

- static void SaveAsSVG (this Page page, string fileName, TextOptions textOption=TextOptions.SubsetFonts, Dictionary< string, string > linkDestinations=null, FilterOption filterOption=default, bool useStyles=false)

  Render the page to an SVG file.
- static void SaveAsSVG (this Page page, Stream stream, TextOptions textOption=TextOptions.SubsetFonts, Dictionary< string, string > linkDestinations=null, FilterOption filterOption=default, bool useStyles=false)

  Render the page to an SVG stream.
- static XmlDocument SaveAsSVG (this Page page, TextOptions textOption=TextOptions.SubsetFonts, Dictionary< string, string > linkDestinations=null, FilterOption filterOption=default, bool useStyles=false)

  Render the page to an SVG document.
- static XmlDocument SaveAsAnimatedSVG (this Animation animation, bool includeControls=false, double durationScaling=1, TextOptions textOption=TextOptions.SubsetFonts, Dictionary< string, string > linkDestinations=null, FilterOption filterOption=default)

  Render the animation to an SVG document, using SVG animations.
- static void SaveAsAnimatedSVG (this Animation animation, Stream stream, bool includeControls=false, double durationScaling=1, TextOptions textOption=TextOptions.SubsetFonts, Dictionary< string, string > linkDestinations=null, FilterOption filterOption=default)

  Render the animation to an SVG stream, using SVG animations.
- static void SaveAsAnimatedSVG (this Animation animation, string fileName, bool includeControls=false, double durationScaling=1, TextOptions textOption=TextOptions.SubsetFonts, Dictionary< string, string > linkDestinations=null, FilterOption filterOption=default)

  Render the animation to an SVG file, using SVG animations.
- static XmlDocument SaveAsAnimatedSVGWithFrames (this Animation animation, bool includeControls=false, double frameRate=60, double durationScaling=1, TextOptions textOption=TextOptions.SubsetFonts, Dictionary< string, string > linkDestinations=null, FilterOption filterOption=default)

  Render the animation to an SVG document, encoding discrete frames.
- static void SaveAsAnimatedSVGWithFrames (this Animation animation, Stream stream, bool includeControls=false, double frameRate=60, double durationScaling=1, TextOptions textOption=TextOptions.SubsetFonts, Dictionary< string, string > linkDestinations=null, FilterOption filterOption=default)

  Render the animation to an SVG stream, encoding discrete frames.
- static void SaveAsAnimatedSVGWithFrames (this Animation animation, string fileName, bool includeControls=false, double frameRate=60, double durationScaling=1, TextOptions textOption=TextOptions.SubsetFonts, Dictionary< string, string > linkDestinations=null, FilterOption filterOption=default)

  Render the animation to an SVG file, encoding discrete frames.
- static void WriteSVGXML (this XmlDocument svgDocument, Stream output)

  Given an XmlDocument that contains an SVG image, write its contents to the specified output Stream.
- static void WriteSVGXML (this XmlDocument svgDocument, string outputFile)

  Given an XmlDocument that contains an SVG image, write its contents to the specified output file.
- static string WriteSVGXML (this XmlDocument svgDocument)

  Given an XmlDocument that contains an SVG image, write its contents to a `string`.

7.173.1 Detailed Description

Contains methods to render a Page as an SVG file.

Definition at line 2324 of file SVGContext.cs.

7.173.2 Member Enumeration Documentation

7.173.2.1 TextOptions

```
enum VectSharp.SVG.SVGContextInterpreter.TextOptions
```

Defines whether the used fonts should be included in the file.

Definition at line 2347 of file SVGContext.cs.

7.173.3 Member Function Documentation

7.173.3.1 SaveAsAnimatedSVG() [1/3]

```
static XmlDocument VectSharp.SVG.SVGContextInterpreter.SaveAsAnimatedSVG (
            this Animation animation,
            bool includeControls = false,
            double durationScaling = 1,
            TextOptions textOption = TextOptions.SubsetFonts,
            Dictionary< string, string > linkDestinations = null,
            FilterOption filterOption = default )  [static]
```

Render the animation to an SVG document, using SVG animations.

Parameters

animation	The Animation to render.
includeControls	If this is `true`, the generated SVG file will contain playback controls that use Javascript to play/pause the animation and change the current time.
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
textOption	Defines whether the used fonts should be included in the file.
linkDestinations	A dictionary associating element tags to link targets. If this is provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to the destination specified in the dictionary. If the destination starts with a hash (#), it is interpreted as the tag of another object in the current document; otherwise, it is interpreted as an external URI.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

An XmlDocument containing the animated SVG image.

Definition at line 2920 of file SVGContext.cs.

7.173.3.2 SaveAsAnimatedSVG() [2/3]

```
static void VectSharp.SVG.SVGContextInterpreter.SaveAsAnimatedSVG (
            this Animation animation,
```

```
            Stream stream,
            bool includeControls = false,
            double durationScaling = 1,
            TextOptions textOption = TextOptions.SubsetFonts,
            Dictionary< string, string > linkDestinations = null,
            FilterOption filterOption = default )  [static]
```

Render the animation to an SVG stream, using SVG animations.

Parameters

animation	The Animation to render.
stream	The Stream on which the SVG document will be written.
includeControls	If this is `true`, the generated SVG file will contain playback controls that use Javascript to play/pause the animation and change the current time.
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
textOption	Defines whether the used fonts should be included in the file.
linkDestinations	A dictionary associating element tags to link targets. If this is provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to the destination specified in the dictionary. If the destination starts with a hash (#), it is interpreted as the tag of another object in the current document; otherwise, it is interpreted as an external URI.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Definition at line 3549 of file SVGContext.cs.

7.173.3.3 SaveAsAnimatedSVG() [3/3]

```
static void VectSharp.SVG.SVGContextInterpreter.SaveAsAnimatedSVG (
            this Animation animation,
            string fileName,
            bool includeControls = false,
            double durationScaling = 1,
            TextOptions textOption = TextOptions.SubsetFonts,
            Dictionary< string, string > linkDestinations = null,
            FilterOption filterOption = default )  [static]
```

Render the animation to an SVG file, using SVG animations.

Parameters

animation	The Animation to render.
fileName	The output file that will be created.
includeControls	If this is `true`, the generated SVG file will contain playback controls that use Javascript to play/pause the animation and change the current time.
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
textOption	Defines whether the used fonts should be included in the file.
linkDestinations	A dictionary associating element tags to link targets. If this is provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to the destination specified in the dictionary. If the destination starts with a hash (#), it is interpreted as the tag of another object in the current document; otherwise, it is interpreted as an external URI.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Definition at line 3565 of file SVGContext.cs.

7.173.3.4 SaveAsAnimatedSVGWithFrames() [1/3]

```
static XmlDocument VectSharp.SVG.SVGContextInterpreter.SaveAsAnimatedSVGWithFrames (
            this Animation animation,
            bool includeControls = false,
            double frameRate = 60,
            double durationScaling = 1,
            TextOptions textOption = TextOptions.SubsetFonts,
            Dictionary< string, string > linkDestinations = null,
            FilterOption filterOption = default )  [static]
```

Render the animation to an SVG document, encoding discrete frames.

Parameters

animation	The Animation to render.
includeControls	If this is `true`, the generated SVG file will contain playback controls that use Javascript to play/pause the animation and change the current time.
frameRate	The target frame rate of the animation, in frames-per-second (fps).
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
textOption	Defines whether the used fonts should be included in the file.
linkDestinations	A dictionary associating element tags to link targets. If this is provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to the destination specified in the dictionary. If the destination starts with a hash (#), it is interpreted as the tag of another object in the current document; otherwise, it is interpreted as an external URI.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Returns

An XmlDocument containing the animated SVG image.

Definition at line 3584 of file SVGContext.cs.

7.173.3.5 SaveAsAnimatedSVGWithFrames() [2/3]

```
static void VectSharp.SVG.SVGContextInterpreter.SaveAsAnimatedSVGWithFrames (
            this Animation animation,
            Stream stream,
            bool includeControls = false,
            double frameRate = 60,
            double durationScaling = 1,
            TextOptions textOption = TextOptions.SubsetFonts,
            Dictionary< string, string > linkDestinations = null,
            FilterOption filterOption = default )  [static]
```

Render the animation to an SVG stream, encoding discrete frames.

Parameters

animation	The Animation to render.
includeControls	If this is `true`, the generated SVG file will contain playback controls that use Javascript to play/pause the animation and change the current time.
stream	The Stream on which the SVG document will be written.
frameRate	The target frame rate of the animation, in frames-per-second (fps).
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
textOption	Defines whether the used fonts should be included in the file.
linkDestinations	A dictionary associating element tags to link targets. If this is provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to the destination specified in the dictionary. If the destination starts with a hash (#), it is interpreted as the tag of another object in the current document; otherwise, it is interpreted as an external URI.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Definition at line 3778 of file SVGContext.cs.

7.173.3.6 SaveAsAnimatedSVGWithFrames() [3/3]

```
static void VectSharp.SVG.SVGContextInterpreter.SaveAsAnimatedSVGWithFrames (
            this Animation animation,
            string fileName,
            bool includeControls = false,
            double frameRate = 60,
            double durationScaling = 1,
            TextOptions textOption = TextOptions.SubsetFonts,
            Dictionary< string, string > linkDestinations = null,
            FilterOption filterOption = default )  [static]
```

Render the animation to an SVG file, encoding discrete frames.

Parameters

animation	The Animation to render.
includeControls	If this is `true`, the generated SVG file will contain playback controls that use Javascript to play/pause the animation and change the current time.
fileName	The output file that will be created.
frameRate	The target frame rate of the animation, in frames-per-second (fps).
durationScaling	A scaling factor that will be applied to all durations in the animation. Values greater than 1 slow down the animation, values smaller than 1 accelerate it. Note that this does not affect the frame rate of the animation.
textOption	Defines whether the used fonts should be included in the file.
linkDestinations	A dictionary associating element tags to link targets. If this is provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to the destination specified in the dictionary. If the destination starts with a hash (#), it is interpreted as the tag of another object in the current document; otherwise, it is interpreted as an external URI.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.

Definition at line 3795 of file SVGContext.cs.

7.173.3.7 SaveAsSVG() [1/3]

```
static void VectSharp.SVG.SVGContextInterpreter.SaveAsSVG (
            this Page page,
            Stream stream,
            TextOptions textOption = TextOptions.SubsetFonts,
            Dictionary< string, string > linkDestinations = null,
            FilterOption filterOption = default,
            bool useStyles = false )  [static]
```

Render the page to an SVG stream.

Parameters

page	The Page to render.
stream	The stream to which the SVG data will be written.
textOption	Defines whether the used fonts should be included in the file.
linkDestinations	A dictionary associating element tags to link targets. If this is provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to the destination specified in the dictionary. If the destination starts with a hash (#), it is interpreted as the tag of another object in the current document; otherwise, it is interpreted as an external URI.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.
useStyles	If this is `false`, presentation attributes are set as attributes on SVG elements. If this is `true`, CSS classes are used to set presentation attributes.

Definition at line 2459 of file SVGContext.cs.

7.173.3.8 SaveAsSVG() [2/3]

```
static void VectSharp.SVG.SVGContextInterpreter.SaveAsSVG (
            this Page page,
            string fileName,
            TextOptions textOption = TextOptions.SubsetFonts,
            Dictionary< string, string > linkDestinations = null,
            FilterOption filterOption = default,
            bool useStyles = false )  [static]
```

Render the page to an SVG file.

Parameters

page	The Page to render.
fileName	The full path to the file to save. If it exists, it will be overwritten.
textOption	Defines whether the used fonts should be included in the file.

Parameters

linkDestinations	A dictionary associating element tags to link targets. If this is provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to the destination specified in the dictionary. If the destination starts with a hash (#), it is interpreted as the tag of another object in the current document; otherwise, it is interpreted as an external URI.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.
useStyles	If this is `false`, presentation attributes are set as attributes on SVG elements. If this is `true`, CSS classes are used to set presentation attributes.

Definition at line 2336 of file SVGContext.cs.

7.173.3.9 SaveAsSVG() [3/3]

```
static XmlDocument VectSharp.SVG.SVGContextInterpreter.SaveAsSVG (
            this Page page,
            TextOptions textOption = TextOptions.SubsetFonts,
            Dictionary< string, string > linkDestinations = null,
            FilterOption filterOption = default,
            bool useStyles = false )  [static]
```

Render the page to an SVG document.

Parameters

page	The Page to render.
textOption	Defines whether the used fonts should be included in the file.
linkDestinations	A dictionary associating element tags to link targets. If this is provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to the destination specified in the dictionary. If the destination starts with a hash (#), it is interpreted as the tag of another object in the current document; otherwise, it is interpreted as an external URI.
filterOption	Defines how and whether image filters should be rasterised when rendering the image.
useStyles	If this is `false`, presentation attributes are set as attributes on SVG elements. If this is `true`, CSS classes are used to set presentation attributes.

Returns

An XmlDocument containing the rendered SVG image.

Definition at line 2474 of file SVGContext.cs.

7.173.3.10 WriteSVGXML() [1/3]

```
static string VectSharp.SVG.SVGContextInterpreter.WriteSVGXML (
            this XmlDocument svgDocument )  [static]
```

Given an XmlDocument that contains an SVG image, write its contents to a `string`.

Parameters

svgDocument	An XmlDocument containing an SVG image

Compared to the built-in methods to save XML documents, this method takes care of some quirks of the SVG format.

Definition at line 4071 of file SVGContext.cs.

7.173.3.11 WriteSVGXML() [2/3]

```
static void VectSharp.SVG.SVGContextInterpreter.WriteSVGXML (
            this XmlDocument svgDocument,
            Stream output )  [static]
```

Given an XmlDocument that contains an SVG image, write its contents to the specified *output* Stream.

Parameters

svgDocument	An XmlDocument containing an SVG image
output	The Stream on which the document will be written.

Compared to the built-in methods to save XML documents, this method takes care of some quirks of the SVG format.

Definition at line 4047 of file SVGContext.cs.

7.173.3.12 WriteSVGXML() [3/3]

```
static void VectSharp.SVG.SVGContextInterpreter.WriteSVGXML (
            this XmlDocument svgDocument,
            string outputFile )  [static]
```

Given an XmlDocument that contains an SVG image, write its contents to the specified output file.

Parameters

svgDocument	An XmlDocument containing an SVG image
outputFile	The file to which the XML document will be saved.

Compared to the built-in methods to save XML documents, this method takes care of some quirks of the SVG format.

Definition at line 4058 of file SVGContext.cs.

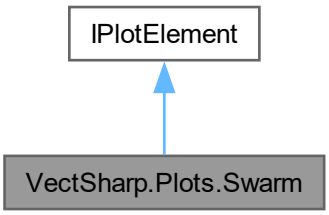
The documentation for this class was generated from the following file:

- VectSharp.SVG/SVGContext.cs

7.174 VectSharp.Plots.Swarm Class Reference

A plot element that draws a swarm plot.

Inheritance diagram for VectSharp.Plots.Swarm:



Public Member Functions

- Swarm (IReadOnlyList< double > position, IReadOnlyList< double > direction, IEnumerable< double > data, IContinuousInvertibleCoordinateSystem coordinateSystem)

  Create a new Swarm instance.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IReadOnlyList< double > Position `[get, set]`

  The position of the origin of the swarm (e.g., the 0 in data space coordinates).
- IReadOnlyList< double > Direction `[get, set]`

  The direction along which the swarm is drawn, in data space coordinates.
- IReadOnlyList< double > Data `[get]`

  The values whose distribution is displayed by the swarm plot.
- double PointSize = 2 `[get, set]`

  The size of the points in the swarm plot, in plot coordinates.
- double PointMargin = 0.25 `[get, set]`

  The minimum margin between points in the swarm plot, in plot coordinates.
- IDataPointElement SwarmPointElement = new PathDataPointElement() `[get, set]`

  The IDataPointElement used to draw the swarm points.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() { Fill = Colours.White } `[get, set]`

  Presentation attributes for the swarm plot.

- IContinuousInvertibleCoordinateSystem CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.
- string Tag `[get, set]`

  A tag to identify the swarm in the plot.

7.174.1 Detailed Description

A plot element that draws a swarm plot.

Definition at line 27 of file Swarm.cs.

7.174.2 Constructor & Destructor Documentation

7.174.2.1 Swarm()

```
VectSharp.Plots.Swarm.Swarm (
            IReadOnlyList< double > position,
            IReadOnlyList< double > direction,
            IEnumerable< double > data,
            IContinuousInvertibleCoordinateSystem coordinateSystem )
```

Create a new Swarm instance.

Parameters

position	The position of the origin of the swarm (e.g., the 0 in data space coordinates).
direction	The direction along which the swarm is drawn, in data space coordinates.
data	The values whose distribution is displayed by the swarm plot.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 82 of file Swarm.cs.

7.174.3 Member Function Documentation

7.174.3.1 Plot()

```
void VectSharp.Plots.Swarm.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 155 of file Swarm.cs.

7.174.4 Property Documentation

7.174.4.1 CoordinateSystem

```
IContinuousInvertibleCoordinateSystem VectSharp.Plots.Swarm.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 67 of file Swarm.cs.

7.174.4.2 Data

```
IReadOnlyList<double> VectSharp.Plots.Swarm.Data  [get]
```

The values whose distribution is displayed by the swarm plot.

Definition at line 42 of file Swarm.cs.

7.174.4.3 Direction

```
IReadOnlyList<double> VectSharp.Plots.Swarm.Direction  [get], [set]
```

The direction along which the swarm is drawn, in data space coordinates.

Definition at line 37 of file Swarm.cs.

7.174.4.4 PointMargin

```
double VectSharp.Plots.Swarm.PointMargin = 0.25  [get], [set]
```

The minimum margin between points in the swarm plot, in plot coordinates.

Definition at line 52 of file Swarm.cs.

7.174.4.5 PointSize

```
double VectSharp.Plots.Swarm.PointSize = 2  [get], [set]
```

The size of the points in the swarm plot, in plot coordinates.

Definition at line 47 of file Swarm.cs.

7.174.4.6 Position

```
IReadOnlyList<double> VectSharp.Plots.Swarm.Position  [get], [set]
```

The position of the origin of the swarm (e.g., the 0 in data space coordinates).

Definition at line 32 of file Swarm.cs.

7.174.4.7 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.Swarm.PresentationAttributes = new PlotElementPresentationAt
{ Fill = Colours.White }  [get], [set]
```

Presentation attributes for the swarm plot.

Definition at line 62 of file Swarm.cs.

7.174.4.8 SwarmPointElement

```
IDataPointElement VectSharp.Plots.Swarm.SwarmPointElement = new PathDataPointElement()  [get],
[set]
```

The IDataPointElement used to draw the swarm points.

Definition at line 57 of file Swarm.cs.

7.174.4.9 Tag

```
string VectSharp.Plots.Swarm.Tag  [get], [set]
```

A tag to identify the swarm in the plot.

Definition at line 73 of file Swarm.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/Swarm.cs

7.175 VectSharp.Markdown.SyntaxHighlighter Class Reference

Contains methods to perform syntax highlighting.

Static Public Member Functions

- static List< List< FormattedString > > GetSyntaxHighlightedLines (string sourceCode, string language)
  Performs syntax highlighting for a specified language on some source code.

7.175.1 Detailed Description

Contains methods to perform syntax highlighting.

Definition at line 73 of file SyntaxHighlighting.cs.

7.175.2 Member Function Documentation

7.175.2.1 GetSyntaxHighlightedLines()

```
static List< List< FormattedString > > VectSharp.Markdown.SyntaxHighlighter.GetSyntaxHighlighted↩
Lines (
            string sourceCode,
            string language )  [static]
```

Performs syntax highlighting for a specified language on some source code.

Parameters

sourceCode	The source code to be highlighted.
language	The name of the language to use for the highlighting.

Returns

A list of lists of FormattedStrings. Each list of FormattedStrings represents a line.

Definition at line 129 of file SyntaxHighlighting.cs.

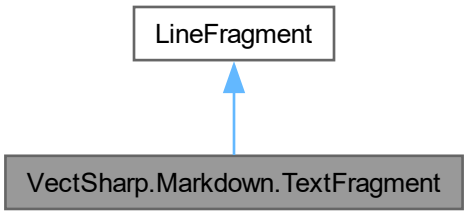
The documentation for this class was generated from the following file:

- VectSharp.Markdown/SyntaxHighlighting.cs

7.176 VectSharp.Markdown.TextFragment Class Reference

A fragment containing text.

Inheritance diagram for VectSharp.Markdown.TextFragment:



Public Member Functions

- TextFragment (Point position, string text, Font font, Colour colour, string tag)

  Create a new TextFragment.
- override void Translate (double deltaX, double deltaY)

  Applies the specified translation to the line fragment.

Parameters

deltaX	*The translation in the horizontal direction.*
deltaY	*The translation in the vertical direction.*

- override double GetAscent (double lineAscent)

  Get the ascent of the LineFragment.

Parameters

lineAscent	*The ascent of the Line on which the fragment should be drawn.*

Returns

The ascent of the LineFragment.

- override double GetMaxY ()

  Get the vertical coordinate of the bottom of the LineFragment.

  Returns

  The vertical coordinates of the bottom of the LineFragment.

- override double GetMaxX ()

  Get the horizontal coordinate of the right side of the LineFragment.

  Returns

  The horizontal coordinate of the right side of the LineFragment.

- override? Point Render (Graphics graphics, double deltaY)

  Render the LineFragment on the specified graphics surface.

Parameters

graphics	*The Graphics surface on which the LineFragment will be drawn.*
deltaY	*Additional translation on the vertical axis.*

Returns

If a text element has been drawn, this method should return the bottom-right corner of the drawn text element, otherwise `null`. This is used to update the MarkdownContext.BottomRight property.

Properties

- string Text `[get]`

  The text contained in the fragment.

- Point Position `[get]`

  The position of the text, relative to the cursor at the start of the line.

- Font Font `[get]`

  The font used to draw the text.

- Colour Colour `[get]`

  The colour used to draw the text.

7.176.1 Detailed Description

A fragment containing text.

Definition at line 71 of file Line.cs.

7.176.2 Constructor & Destructor Documentation

7.176.2.1 TextFragment()

```
VectSharp.Markdown.TextFragment.TextFragment (
            Point position,
            string text,
            Font font,
            Colour colour,
            string tag )
```

Create a new TextFragment.

Parameters

position	The position of the text.
text	The text contained in the fragment.
font	The font used to draw the text.
colour	The colour used to draw the text.
tag	Graphics action tag for the text.

Definition at line 101 of file Line.cs.

7.176.3 Member Function Documentation

7.176.3.1 GetAscent()

```
override double VectSharp.Markdown.TextFragment.GetAscent (
            double lineAscent )  [virtual]
```

Get the ascent of the LineFragment.

Parameters

lineAscent	The ascent of the Line on which the fragment should be drawn.

Returns

The ascent of the LineFragment.

Implements VectSharp.Markdown.LineFragment.

Definition at line 117 of file Line.cs.

7.176.3.2 GetMaxX()

```
override double VectSharp.Markdown.TextFragment.GetMaxX ( )  [virtual]
```

Get the horizontal coordinate of the right side of the LineFragment.

Returns

The horizontal coordinate of the right side of the LineFragment.

Implements VectSharp.Markdown.LineFragment.

Definition at line 129 of file Line.cs.

7.176.3.3 GetMaxY()

```
override double VectSharp.Markdown.TextFragment.GetMaxY ( )  [virtual]
```

Get the vertical coordinate of the bottom of the LineFragment.

Returns

The vertical coordinates of the bottom of the LineFragment.

Implements VectSharp.Markdown.LineFragment.

Definition at line 123 of file Line.cs.

7.176.3.4 Render()

```
override?  Point VectSharp.Markdown.TextFragment.Render (
             Graphics graphics,
             double deltaY )  [virtual]
```

Render the LineFragment on the specified *graphics* surface.

Parameters

graphics	The Graphics surface on which the LineFragment will be drawn.
deltaY	Additional translation on the vertical axis.

Returns

If a text element has been drawn, this method should return the bottom-right corner of the drawn text element, otherwise `null`. This is used to update the MarkdownContext.BottomRight property.

Implements VectSharp.Markdown.LineFragment.

Definition at line 136 of file Line.cs.

7.176.3.5 Translate()

```
override void VectSharp.Markdown.TextFragment.Translate (
             double deltaX,
             double deltaY )  [virtual]
```

Applies the specified translation to the line fragment.

Parameters

deltaX	The translation in the horizontal direction.
deltaY	The translation in the vertical direction.

Implements VectSharp.Markdown.LineFragment.

Definition at line 111 of file Line.cs.

7.176.4 Property Documentation

7.176.4.1 Colour

```
Colour VectSharp.Markdown.TextFragment.Colour  [get]
```

The colour used to draw the text.

Definition at line 91 of file Line.cs.

7.176.4.2 Font

```
Font VectSharp.Markdown.TextFragment.Font  [get]
```

The font used to draw the text.

Definition at line 86 of file Line.cs.

7.176.4.3 Position

```
Point VectSharp.Markdown.TextFragment.Position  [get]
```

The position of the text, relative to the cursor at the start of the line.

Definition at line 81 of file Line.cs.

7.176.4.4 Text

```
string VectSharp.Markdown.TextFragment.Text  [get]
```

The text contained in the fragment.

Definition at line 76 of file Line.cs.

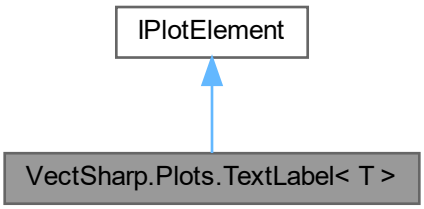
The documentation for this class was generated from the following file:

- VectSharp.Markdown/Line.cs

7.177 VectSharp.Plots.TextLabel< T > Class Template Reference

A plot element that draws a single text label.

Inheritance diagram for VectSharp.Plots.TextLabel< T >:



Public Member Functions

- TextLabel (string label, T position, ICoordinateSystem< T > coordinateSystem)

  Create a new TextLabel< T > instance.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- T Position `[get, set]`

  The position of the label, in data space coordinates.
- string Label `[get, set]`

  The text of the label. This can include formatting specifiers (see the documentation for the FormattedText.↩ Format(string, Font, Font, Font, Font, Brush) method).
- double Rotation = 0 `[get, set]`

  The angle at which the text is drawn, with respect to the horizontal.
- TextBaselines Baseline = TextBaselines.Middle `[get, set]`

  The baseline for the text.
- TextAnchors Alignment = TextAnchors.Center `[get, set]`

  The alignment for the text.
- ICoordinateSystem< T > CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() { Stroke = null } `[get, set]`

  Presentation attributes determining the appearance (stroke and fill colour, etc.) of the text label.
- string Tag `[get, set]`

  A tag to identify the label in the plot.

7.177.1 Detailed Description

A plot element that draws a single text label.

Template Parameters

T	The kind of data describing the data points (generally, `IReadOnlyList<double>`).

Definition at line 226 of file DataPoints.cs.

7.177.2 Constructor & Destructor Documentation

7.177.2.1 TextLabel()

```
VectSharp.Plots.TextLabel< T >.TextLabel (
            string label,
            T position,
            ICoordinateSystem< T > coordinateSystem )
```

Create a new TextLabel<T> instance.

Parameters

label	The text of the label. This can include formatting specifiers (see the documentation for the FormattedText.Format(string, Font, Font, Font, Font, Brush) method).
position	The position of the label, in data space coordinates.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 275 of file DataPoints.cs.

7.177.3 Member Function Documentation

7.177.3.1 Plot()

```
void VectSharp.Plots.TextLabel< T >.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 283 of file DataPoints.cs.

7.177.4 Property Documentation

7.177.4.1 Alignment

```
TextAnchors VectSharp.Plots.TextLabel< T >.Alignment = TextAnchors.Center  [get], [set]
```

The alignment for the text.

Definition at line 251 of file DataPoints.cs.

7.177.4.2 Baseline

```
TextBaselines VectSharp.Plots.TextLabel< T >.Baseline = TextBaselines.Middle  [get], [set]
```

The baseline for the text.

Definition at line 246 of file DataPoints.cs.

7.177.4.3 CoordinateSystem

```
ICoordinateSystem<T> VectSharp.Plots.TextLabel< T >.CoordinateSystem  [get], [set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 256 of file DataPoints.cs.

7.177.4.4 Label

```
string VectSharp.Plots.TextLabel< T >.Label  [get], [set]
```

The text of the label. This can include formatting specifiers (see the documentation for the FormattedText.↩ Format(string, Font, Font, Font, Font, Brush) method).

Definition at line 236 of file DataPoints.cs.

7.177.4.5 Position

```
T VectSharp.Plots.TextLabel< T >.Position  [get], [set]
```

The position of the label, in data space coordinates.

Definition at line 231 of file DataPoints.cs.

7.177.4.6 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.TextLabel< T >.PresentationAttributes = new
PlotElementPresentationAttributes() { Stroke = null }  [get], [set]
```

Presentation attributes determining the appearance (stroke and fill colour, etc.) of the text label.

Definition at line 262 of file DataPoints.cs.

7.177.4.7 Rotation

```
double VectSharp.Plots.TextLabel< T >.Rotation = 0  [get], [set]
```

The angle at which the text is drawn, with respect to the horizontal.

Definition at line 241 of file DataPoints.cs.

7.177.4.8 Tag

```
string VectSharp.Plots.TextLabel< T >.Tag  [get], [set]
```

A tag to identify the label in the plot.

Definition at line 267 of file DataPoints.cs.

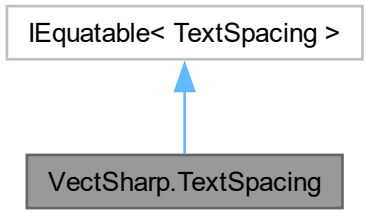
The documentation for this class was generated from the following file:

- VectSharp.Plots/DataPoints.cs

7.178 VectSharp.TextSpacing Struct Reference

Contains information about text spacing.

Inheritance diagram for VectSharp.TextSpacing:



Public Member Functions

- TextSpacing (double nonWhitespaceScale, double nonWhitespaceAdd, double whitespaceScale, double whitespaceAdd)

  Create a new TextSpacing with the specified spacing parameters.
- TextSpacing (double scale, double add)

  Create a new TextSpacing with the specified spacing parameters.
- bool Equals (TextSpacing other)
- override bool Equals (object obj)
- override int GetHashCode ()

Static Public Member Functions

- static bool operator== (TextSpacing left, TextSpacing right)

  Compares two TextSpacing objects.
- static bool operator!= (TextSpacing left, TextSpacing right)

  Compares two TextSpacing objects.

Properties

- double WhitespaceScale `[get]`

  Scaling factor for whitespace characters.
- double WhitespaceAdd `[get]`

  Increment for whitespace characters.
- double NonWhitespaceScale `[get]`

  Scaling factor for non-whitespace characters.
- double NonWhitespaceAdd `[get]`

  Increment for non-whitespace characters.
- static TextSpacing Default = new TextSpacing(1, 0) `[get]`

  Default text spacing.

7.178.1 Detailed Description

Contains information about text spacing.

Definition at line 74 of file Enums.cs.

7.178.2 Constructor & Destructor Documentation

7.178.2.1 TextSpacing() [1/2]

```
VectSharp.TextSpacing.TextSpacing (
            double nonWhitespaceScale,
            double nonWhitespaceAdd,
            double whitespaceScale,
            double whitespaceAdd )
```

Create a new TextSpacing with the specified spacing parameters.

Parameters

nonWhitespaceScale	Scaling factor for non-whitespace characters.
nonWhitespaceAdd	Increment for non-whitespace characters.
whitespaceScale	Scaling factor for whitespace characters.
whitespaceAdd	Increment for whitespace characters.

Definition at line 104 of file Enums.cs.

7.178.2.2 TextSpacing() [2/2]

```
VectSharp.TextSpacing.TextSpacing (
            double scale,
            double add )
```

Create a new TextSpacing with the specified spacing parameters.

Parameters

scale	Scaling factor.
add	Increment.

Definition at line 117 of file Enums.cs.

7.178.3 Member Function Documentation

7.178.3.1 Equals() [1/2]

```
override bool VectSharp.TextSpacing.Equals (
            object obj )
```

Definition at line 137 of file Enums.cs.

7.178.3.2 Equals() [2/2]

```
bool VectSharp.TextSpacing.Equals (
            TextSpacing other )
```

Definition at line 131 of file Enums.cs.

7.178.3.3 GetHashCode()

```
override int VectSharp.TextSpacing.GetHashCode ( )
```

Definition at line 150 of file Enums.cs.

7.178.3.4 operator"!=()

```
static bool VectSharp.TextSpacing.operator!= (
            TextSpacing left,
            TextSpacing right )  [static]
```

Compares two TextSpacing objects.

Parameters

left	The first TextSpacing object.
right	The second TextSpacing object.

Returns

`false` if the two TextSpacing objects specify the same spacing parameters, `true` otherwise.

Definition at line 180 of file Enums.cs.

7.178.3.5 operator==()

```
static bool VectSharp.TextSpacing.operator== (
            TextSpacing left,
            TextSpacing right )  [static]
```

Compares two TextSpacing objects.

Parameters

left	The first TextSpacing object.
right	The second TextSpacing object.

Returns

`true` if the two TextSpacing objects specify the same spacing parameters, `false` otherwise.

Definition at line 169 of file Enums.cs.

7.178.4 Property Documentation

7.178.4.1 Default

```
TextSpacing VectSharp.TextSpacing.Default = new TextSpacing(1, 0)  [static], [get]
```

Default text spacing.

Definition at line 128 of file Enums.cs.

7.178.4.2 NonWhitespaceAdd

```
double VectSharp.TextSpacing.NonWhitespaceAdd  [get]
```

Increment for non-whitespace characters.

Definition at line 95 of file Enums.cs.

7.178.4.3 NonWhitespaceScale

```
double VectSharp.TextSpacing.NonWhitespaceScale  [get]
```

Scaling factor for non-whitespace characters.

Definition at line 90 of file Enums.cs.

7.178.4.4 WhitespaceAdd

```
double VectSharp.TextSpacing.WhitespaceAdd [get]
```

Increment for whitespace characters.

Definition at line 84 of file Enums.cs.

7.178.4.5 WhitespaceScale

```
double VectSharp.TextSpacing.WhitespaceScale [get]
```

Scaling factor for whitespace characters.

Definition at line 79 of file Enums.cs.

The documentation for this struct was generated from the following file:

- VectSharp/Enums.cs

7.179 VectSharp.Transition Class Reference

Describes the transition between two successive Frames.

Public Member Functions

- Transition (double duration, IEasing easing=null, Dictionary< string, IEasing > easings=null)

  Creates a new Transition with the specified duration and easings.

Properties

- double Duration [get]

  The duration of the transition, in milliseconds.
- IEasing OverallEasing = null [get]

  The IEasing to apply to all elements for which another easing is not specified. Set to null to use the default linear easing.
- Dictionary< string, IEasing > Easings = null [get]

  A dictionary associating graphic action tags to the corresponding IEasing.

7.179.1 Detailed Description

Describes the transition between two successive Frames.

Definition at line 1371 of file Animation.cs.

7.179.2 Constructor & Destructor Documentation

7.179.2.1 Transition()

```
VectSharp.Transition.Transition (
            double duration,
            IEasing easing = null,
            Dictionary< string, IEasing > easings = null )
```

Creates a new Transition with the specified duration and easings.

Parameters

duration	The duration of the transition, in milliseconds.
easing	The IEasing to apply to all elements for which another easing is not specified. Set to null to use the default linear easing.
easings	A dictionary associating graphic action tags to the corresponding IEasing.

Definition at line 1394 of file Animation.cs.

7.179.3 Property Documentation

7.179.3.1 Duration

```
double VectSharp.Transition.Duration  [get]
```

The duration of the transition, in milliseconds.

Definition at line 1376 of file Animation.cs.

7.179.3.2 Easings

```
Dictionary<string, IEasing> VectSharp.Transition.Easings = null  [get]
```

A dictionary associating graphic action tags to the corresponding IEasing.

Definition at line 1386 of file Animation.cs.

7.179.3.3 OverallEasing

```
IEasing VectSharp.Transition.OverallEasing = null  [get]
```

The IEasing to apply to all elements for which another easing is not specified. Set to null to use the default linear easing.

Definition at line 1381 of file Animation.cs.

The documentation for this class was generated from the following file:

- VectSharp/Animation.cs

7.180 VectSharp.TrueTypeFile Class Reference

Represents a font file in TrueType format. Reference: http://stevehanov.ca/blog/?id=143, https://developer.apple.com/fonts/TrueType-Reference-Manual/, https://docs.microsoft.com/en-us/typography/opentype/spec/

Classes

- struct Bearings

  Represents the left- and right-side bearings of a glyph.
- class PairKerning

  Contains information describing how the position of two glyphs in a kerning pair should be altered.
- struct TrueTypeName

  Represets a TrueType name.
- struct TrueTypePoint

  Represents a point in a TrueType path description.
- struct VerticalMetrics

  Represents the maximum heigth above and depth below the baseline of a glyph.

Public Member Functions

- void Destroy ()

  Remove this TrueType file from the cache, clear the tables and release the FontStream. Only call this when the actual file that was used to create this object needs to be changed!
- TrueTypeFile SubsetFont (string charactersToInclude, bool consolidateAt32=false, Dictionary< char, char > outputEncoding=null)

  Create a subset of the TrueType file, containing only the glyphs for the specified characters.
- string GetFontFamilyName ()

  Obtains the font family name from the TrueType file.
- string GetFullFontFamilyName ()

  Obtains the full font family name from the TrueType file.
- string GetFontName ()

  Obtains the PostScript font name from the TrueType file.
- ushort GetFirstCharIndex ()

  Returns the index of the first character glyph represented by the font.
- ushort GetLastCharIndex ()

  Returns the index of the last character glyph represented by the font.
- bool IsItalic ()

  Determines whether the typeface is Italic or Oblique or not.
- bool IsOblique ()

  Determines whether the typeface is Oblique or not.
- bool IsBold ()

  Determines whether the typeface is Bold or not.
- bool IsFixedPitch ()

  Determines whether the typeface is fixed-pitch (aka monospaces) or not.
- bool IsSerif ()

  Determines whether the typeface is serifed or not.
- bool IsScript ()

  Determines whether the typeface is a script typeface or not.

- int GetGlyphIndex (char glyph)

  Determines the index of the glyph corresponding to a certain character.
- TrueTypePoint[ ][ ] GetGlyphPath (int glyphIndex, double size)

  Get the path that describes the shape of a glyph.
- TrueTypePoint[ ][ ] GetGlyphPath (char glyph, double size)

  Get the path that describes the shape of a glyph.
- double Get1000EmGlyphWidth (char glyph)

  Computes the advance width of a glyph, in thousandths of em unit.
- double Get1000EmGlyphWidth (int glyphIndex)

  Computes the advance width of a glyph, in thousandths of em unit.
- double Get1000EmWinAscent ()

  Computes the font's Win ascent, in thousandths of em unit.
- double Get1000EmAscent ()

  Computes the font ascent, in thousandths of em unit.
- double Get1000EmDescent ()

  Computes the font descent, in thousandths of em unit.
- double Get1000EmYMax ()

  Computes the maximum height over the baseline of the font, in thousandths of em unit.
- double Get1000EmYMin ()

  Computes the maximum depth below the baseline of the font, in thousandths of em unit.
- double Get1000EmXMax ()

  Computes the maximum distance to the right of the glyph origin of the font, in thousandths of em unit.
- double Get1000EmXMin ()

  Computes the maximum distance to the left of the glyph origin of the font, in thousandths of em unit.
- Bearings Get1000EmGlyphBearings (char glyph)

  Computes the left- and right- side bearings of a glyph, in thousandths of em unit.
- VerticalMetrics Get1000EmGlyphVerticalMetrics (char glyph)

  Computes the vertical metrics of a glyph, in thousandths of em unit.
- double Get1000EmUnderlinePosition ()

  Computes the distance of the top of the underline from the baseline, in thousandths of em unit.
- double Get1000EmUnderlineThickness ()

  Computes the thickness of the underline, in thousandths of em unit.
- int GetUnitsPerEm ()

  Returns the number of units for each em in the font file. Multiplying any glyph measurement by this size should generally lead to an integer value.
- double GetItalicAngle ()

  Computes the italic angle for the current font, in counter-clockwise degrees from the vertical. This is negative for text that leans forwards.
- double[ ] Get1000EmUnderlineIntersections (char glyph, double position, double thickness)

  Computes the intersections between an underline at the specified position and thickness and a glyph, in thousandths of em units.
- PairKerning Get1000EmKerning (char glyph1, char glyph2)

  Gets the kerning between two glyphs.
- PairKerning Get1000EmKerning (int glyph1Index, int glyph2Index)

  Gets the kerning between two glyphs.

Static Public Member Functions

- static bool GetNames (string file, out List< TrueTypeName > names)

  Determines whether the file is a valid TrueType file and retrieves the list of names defined in it.
- static bool GetNames (Stream fs, out List< TrueTypeName > names)

  Determines whether the stream contains a valid TrueType file and retrieves the list of names defined in it.

Properties

- Stream FontStream [get]

  A stream pointing to the TrueType file source (either on disk or in memory). Never dispose this stream directly; if you really need to, call Destroy instead.

7.180.1 Detailed Description

Represents a font file in TrueType format. Reference: http://stevehanov.ca/blog/?id=143, https://developer.apple.com/fonts/TrueType-Reference-Manual/, https://docs.microsoft.com/en-us/typography/opentype/spec/

Definition at line 30 of file TrueType.cs.

7.180.2 Member Function Documentation

7.180.2.1 Destroy()

```
void VectSharp.TrueTypeFile.Destroy ( )
```

Remove this TrueType file from the cache, clear the tables and release the FontStream. Only call this when the actual file that was used to create this object needs to be changed!

Definition at line 53 of file TrueType.cs.

7.180.2.2 Get1000EmAscent()

```
double VectSharp.TrueTypeFile.Get1000EmAscent ( )
```

Computes the font ascent, in thousandths of em unit.

Returns

The font ascent in thousandths of em unit.

Definition at line 2267 of file TrueType.cs.

7.180.2.3 Get1000EmDescent()

```
double VectSharp.TrueTypeFile.Get1000EmDescent ( )
```

Computes the font descent, in thousandths of em unit.

Returns

The font descent in thousandths of em unit.

Definition at line 2277 of file TrueType.cs.

7.180.2.4 Get1000EmGlyphBearings()

```
Bearings VectSharp.TrueTypeFile.Get1000EmGlyphBearings (
            char glyph )
```

Computes the left- and right- side bearings of a glyph, in thousandths of em unit.

Parameters

glyph	The glyph whose bearings are to be computed.

Returns

The left- and right- side bearings of the glyph in thousandths of em unit

Definition at line 2359 of file TrueType.cs.

7.180.2.5 Get1000EmGlyphVerticalMetrics()

```
VerticalMetrics VectSharp.TrueTypeFile.Get1000EmGlyphVerticalMetrics (
            char glyph )
```

Computes the vertical metrics of a glyph, in thousandths of em unit.

Parameters

glyph	The glyph whose vertical metrics are to be computed.

Returns

The vertical metrics of a glyph, in thousandths of em unit.

Definition at line 2407 of file TrueType.cs.

7.180.2.6 Get1000EmGlyphWidth() [1/2]

```
double VectSharp.TrueTypeFile.Get1000EmGlyphWidth (
            char glyph )
```

Computes the advance width of a glyph, in thousandths of em unit.

Parameters

glyph	The glyph whose advance width is to be computed.

Returns

The advance width of the glyph in thousandths of em unit.

Definition at line 2218 of file TrueType.cs.

7.180.2.7 Get1000EmGlyphWidth() [2/2]

```
double VectSharp.TrueTypeFile.Get1000EmGlyphWidth (
            int glyphIndex )
```

Computes the advance width of a glyph, in thousandths of em unit.

Parameters

glyphIndex	The index of the glyph whose advance width is to be computed.

Returns

The advance width of the glyph in thousandths of em unit.

Definition at line 2236 of file TrueType.cs.

7.180.2.8 Get1000EmKerning() [1/2]

```
PairKerning VectSharp.TrueTypeFile.Get1000EmKerning (
            char glyph1,
            char glyph2 )
```

Gets the kerning between two glyphs.

Parameters

glyph1	The first glyph of the kerning pair.
glyph2	The second glyph of the kerning pair.

Returns

A PairKerning object containing information about how the position of each glyphs should be altered.

Definition at line 2636 of file TrueType.cs.

7.180.2.9 Get1000EmKerning() [2/2]

```
PairKerning VectSharp.TrueTypeFile.Get1000EmKerning (
            int glyph1Index,
            int glyph2Index )
```

Gets the kerning between two glyphs.

Parameters

glyph1Index	The index of the first glyph of the kerning pair.
glyph2Index	The index of the second glyph of the kerning pair.

Returns

A PairKerning object containing information about how the position of each glyphs should be altered.

Definition at line 2650 of file TrueType.cs.

7.180.2.10 Get1000EmUnderlineIntersections()

```
double[] VectSharp.TrueTypeFile.Get1000EmUnderlineIntersections (
            char glyph,
            double position,
            double thickness )
```

Computes the intersections between an underline at the specified position and thickness and a glyph, in thousandths of em units.

Parameters

glyph	The glyph whose intersections with the underline will be computed.
position	The distance of the top of the underline from the baseline, in thousandths of em unit.
thickness	The thickness of the underline, in thousandths of em unit.

Returns

If the underline does not intersect the glyph, this method returns `null`. Otherwise, it returns an array containing two elements, representing the horizontal coordinates of the leftmost and rightmost intersection points.

Definition at line 2486 of file TrueType.cs.

7.180.2.11 Get1000EmUnderlinePosition()

```
double VectSharp.TrueTypeFile.Get1000EmUnderlinePosition ( )
```

Computes the distance of the top of the underline from the baseline, in thousandths of em unit.

Returns

The distance of the top of the underline from the baseline, in thousandths of em unit.

Definition at line 2424 of file TrueType.cs.

7.180.2.12 Get1000EmUnderlineThickness()

```
double VectSharp.TrueTypeFile.Get1000EmUnderlineThickness ( )
```

Computes the thickness of the underline, in thousandths of em unit.

Returns

The thickness of the underline, in thousandths of em unit.

Definition at line 2440 of file TrueType.cs.

7.180.2.13 Get1000EmWinAscent()

```
double VectSharp.TrueTypeFile.Get1000EmWinAscent ( )
```

Computes the font's Win ascent, in thousandths of em unit.

Returns

The font's Win ascent in thousandths of em unit.

Definition at line 2247 of file TrueType.cs.

7.180.2.14 Get1000EmXMax()

```
double VectSharp.TrueTypeFile.Get1000EmXMax ( )
```

Computes the maximum distance to the right of the glyph origin of the font, in thousandths of em unit.

Returns

The maximum distance to the right of the glyph origin of the font in thousandths of em unit.

Definition at line 2304 of file TrueType.cs.

7.180.2.15 Get1000EmXMin()

```
double VectSharp.TrueTypeFile.Get1000EmXMin ( )
```

Computes the maximum distance to the left of the glyph origin of the font, in thousandths of em unit.

Returns

The maximum distance to the left of the glyph origin of the font in thousandths of em unit.

Definition at line 2313 of file TrueType.cs.

7.180.2.16 Get1000EmYMax()

```
double VectSharp.TrueTypeFile.Get1000EmYMax ( )
```

Computes the maximum height over the baseline of the font, in thousandths of em unit.

Returns

The maximum height over the baseline of the font in thousandths of em unit.

Definition at line 2286 of file TrueType.cs.

7.180.2.17 Get1000EmYMin()

```
double VectSharp.TrueTypeFile.Get1000EmYMin ( )
```

Computes the maximum depth below the baseline of the font, in thousandths of em unit.

Returns

The maximum depth below the baseline of the font in thousandths of em unit.

Definition at line 2295 of file TrueType.cs.

7.180.2.18 GetFirstCharIndex()

```
ushort VectSharp.TrueTypeFile.GetFirstCharIndex ( )
```

Returns the index of the first character glyph represented by the font.

Returns

The index of the first character glyph represented by the font.

Definition at line 2056 of file TrueType.cs.

7.180.2.19 GetFontFamilyName()

```
string VectSharp.TrueTypeFile.GetFontFamilyName ( )
```

Obtains the font family name from the TrueType file.

Returns

The font family name, if available; `null` otherwise.

Definition at line 1982 of file TrueType.cs.

7.180.2.20 GetFontName()

```
string VectSharp.TrueTypeFile.GetFontName ( )
```

Obtains the PostScript font name from the TrueType file.

Returns

The PostScript font name, if available; `null` otherwise.

Definition at line 2037 of file TrueType.cs.

7.180.2.21 GetFullFontFamilyName()

```
string VectSharp.TrueTypeFile.GetFullFontFamilyName ( )
```

Obtains the full font family name from the TrueType file.

Returns

The full font family name, if available; `null` otherwise.

Definition at line 2009 of file TrueType.cs.

7.180.2.22 GetGlyphIndex()

```
int VectSharp.TrueTypeFile.GetGlyphIndex (
            char glyph )
```

Determines the index of the glyph corresponding to a certain character.

Parameters

glyph	The character whose glyph is sought.

Returns

The index of the glyph in the TrueType file.

Definition at line 2146 of file TrueType.cs.

7.180.2.23 GetGlyphPath() [1/2]

```
TrueTypePoint[][] VectSharp.TrueTypeFile.GetGlyphPath (
            char glyph,
            double size )
```

Get the path that describes the shape of a glyph.

Parameters

glyph	The glyph whose path is sought.
size	The font size to be used for the font coordinates.

Returns

An array of contours, each of which is itself an array of TrueType points.

Definition at line 2208 of file TrueType.cs.

7.180.2.24 GetGlyphPath() [2/2]

```
TrueTypePoint[][] VectSharp.TrueTypeFile.GetGlyphPath (
            int glyphIndex,
            double size )
```

Get the path that describes the shape of a glyph.

Parameters

glyphIndex	The index of the glyph whose path is sought.
size	The font size to be used for the font coordinates.

Returns

An array of contours, each of which is itself an array of TrueType points.

Definition at line 2197 of file TrueType.cs.

7.180.2.25 GetItalicAngle()

```
double VectSharp.TrueTypeFile.GetItalicAngle ( )
```

Computes the italic angle for the current font, in counter-clockwise degrees from the vertical. This is negative for text that leans forwards.

Returns

Definition at line 2466 of file TrueType.cs.

7.180.2.26 GetLastCharIndex()

```
ushort VectSharp.TrueTypeFile.GetLastCharIndex ( )
```

Returns the index of the last character glyph represented by the font.

Returns

The index of the last character glyph represented by the font.

Definition at line 2067 of file TrueType.cs.

7.180.2.27 GetNames() [1/2]

```
static bool VectSharp.TrueTypeFile.GetNames (
            Stream fs,
            out List< TrueTypeName > names )  [static]
```

Determines whether the stream contains a valid TrueType file and retrieves the list of names defined in it.

Parameters

fs	The stream containing the TrueType file.
names	The list of names that will be returned. These will only include names with TrueType identifiers 1, 4, 6, or 16.

Returns

If the stream does not contain a valid TrueType file, `false`. If the TrueType file appears valid but does not contain a name table, `true`, but *names* will be `null`. Otherwise, `true` and *names* will not be null (but it might still be empty).

Definition at line 621 of file TrueType.cs.

7.180.2.28 GetNames() [2/2]

```
static bool VectSharp.TrueTypeFile.GetNames (
            string file,
            out List< TrueTypeName > names )  [static]
```

Determines whether the file is a valid TrueType file and retrieves the list of names defined in it.

Parameters

file	The TrueType file.
names	The list of names that will be returned. These will only include names with TrueType identifiers 1, 4, 6, or 16.

Returns

If an error occurred while opening the file, or the file does not contain a valid TrueType file, `false`. If the TrueType file appears valid but does not contain a name table, `true`, but *names* will be `null`. Otherwise, `true` and *names* will not be null (but it might still be empty).

Definition at line 597 of file TrueType.cs.

7.180.2.29 GetUnitsPerEm()

```
int VectSharp.TrueTypeFile.GetUnitsPerEm ( )
```

Returns the number of units for each em in the font file. Multiplying any glyph measurement by this size should generally lead to an integer value.

Returns

The number of units for each em in the font file.

Definition at line 2457 of file TrueType.cs.

7.180.2.30 IsBold()

```
bool VectSharp.TrueTypeFile.IsBold ( )
```

Determines whether the typeface is Bold or not.

Returns

A bool indicating whether the typeface is Bold or not

Definition at line 2101 of file TrueType.cs.

7.180.2.31 IsFixedPitch()

```
bool VectSharp.TrueTypeFile.IsFixedPitch ( )
```

Determines whether the typeface is fixed-pitch (aka monospaces) or not.

Returns

A bool indicating whether the typeface is fixed-pitch (aka monospaces) or not.

Definition at line 2112 of file TrueType.cs.

7.180.2.32 IsItalic()

```
bool VectSharp.TrueTypeFile.IsItalic ( )
```

Determines whether the typeface is Italic or Oblique or not.

Returns

A bool indicating whether the typeface is Italic or Oblique or not.

Definition at line 2079 of file TrueType.cs.

7.180.2.33 IsOblique()

```
bool VectSharp.TrueTypeFile.IsOblique ( )
```

Determines whether the typeface is Oblique or not.

Returns

A bool indicating whether the typeface is Oblique or not.

Definition at line 2090 of file TrueType.cs.

7.180.2.34 IsScript()

```
bool VectSharp.TrueTypeFile.IsScript ( )
```

Determines whether the typeface is a script typeface or not.

Returns

A bool indicating whether the typeface is a script typeface or not.

Definition at line 2134 of file TrueType.cs.

7.180.2.35 IsSerif()

```
bool VectSharp.TrueTypeFile.IsSerif ( )
```

Determines whether the typeface is serifed or not.

Returns

A bool indicating whether the typeface is serifed or not.

Definition at line 2123 of file TrueType.cs.

7.180.2.36 SubsetFont()

```
TrueTypeFile VectSharp.TrueTypeFile.SubsetFont (
            string charactersToInclude,
            bool consolidateAt32 = false,
            Dictionary< char, char > outputEncoding = null )
```

Create a subset of the TrueType file, containing only the glyphs for the specified characters.

Parameters

charactersToInclude	A string containing the characters for which the glyphs should be included.
consolidateAt32	If true, the character map is rearranged so that the included glyphs start at the unicode U+0032 control point.
outputEncoding	If *consolidateAt32* is true, entries will be added to this dictionary mapping the original characters to the new map (that starts at U+0033).

Returns

The subsetted TrueType file.

Definition at line 698 of file TrueType.cs.

7.180.3 Property Documentation

7.180.3.1 FontStream

```
Stream VectSharp.TrueTypeFile.FontStream  [get]
```

A stream pointing to the TrueType file source (either on disk or in memory). Never dispose this stream directly; if you really need to, call Destroy instead.

Definition at line 47 of file TrueType.cs.

The documentation for this class was generated from the following file:

- VectSharp/TrueType.cs

7.181 VectSharp.TrueTypeFile.TrueTypeName Struct Reference

Represets a TrueType name.

Public Types

- enum NameIdentifier

  Describes a kind of TrueType name.

Public Attributes

- NameIdentifier NameId

  The kind of name represented by this struct.
- string Name

  The name represented by this struct.

7.181.1 Detailed Description

Represets a TrueType name.

Definition at line 544 of file TrueType.cs.

7.181.2 Member Enumeration Documentation

7.181.2.1 NameIdentifier

`enum VectSharp.TrueTypeFile.TrueTypeName.NameIdentifier`

Describes a kind of TrueType name.

Definition at line 549 of file TrueType.cs.

7.181.3 Member Data Documentation

7.181.3.1 Name

`string VectSharp.TrueTypeFile.TrueTypeName.Name`

The name represented by this struct.

Definition at line 580 of file TrueType.cs.

7.181.3.2 NameId

`NameIdentifier VectSharp.TrueTypeFile.TrueTypeName.NameId`

The kind of name represented by this struct.

Definition at line 575 of file TrueType.cs.

The documentation for this struct was generated from the following file:

- VectSharp/TrueType.cs

7.182 VectSharp.TrueTypeFile.TrueTypePoint Struct Reference

Represents a point in a TrueType path description.

Public Attributes

- double X

  The horizontal coordinate of the point.
- double Y

  The vertical coordinate of the point.
- bool IsOnCurve

  Whether the point is a point on the curve, or a control point of a quadratic Bezier curve.

7.182.1 Detailed Description

Represents a point in a TrueType path description.

Definition at line 1496 of file TrueType.cs.

7.182.2 Member Data Documentation

7.182.2.1 IsOnCurve

```
bool VectSharp.TrueTypeFile.TrueTypePoint.IsOnCurve
```

Whether the point is a point on the curve, or a control point of a quadratic Bezier curve.

Definition at line 1511 of file TrueType.cs.

7.182.2.2 X

```
double VectSharp.TrueTypeFile.TrueTypePoint.X
```

The horizontal coordinate of the point.

Definition at line 1501 of file TrueType.cs.

7.182.2.3 Y

```
double VectSharp.TrueTypeFile.TrueTypePoint.Y
```

The vertical coordinate of the point.

Definition at line 1506 of file TrueType.cs.

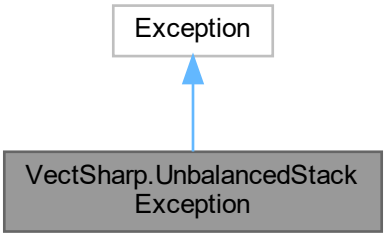
The documentation for this struct was generated from the following file:

- VectSharp/TrueType.cs

7.183 VectSharp.UnbalancedStackException Class Reference

The exception that is thrown when an unbalanced graphics state stack occurs.

Inheritance diagram for VectSharp.UnbalancedStackException:



7.183.1 Detailed Description

The exception that is thrown when an unbalanced graphics state stack occurs.

Definition at line 255 of file Graphics.cs.

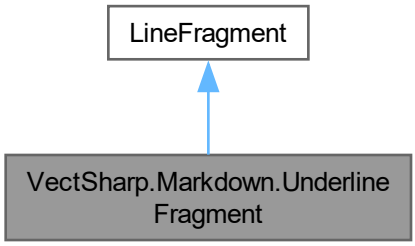
The documentation for this class was generated from the following file:

- VectSharp/Graphics.cs

7.184 VectSharp.Markdown.UnderlineFragment Class Reference

Represents a text underline.

Inheritance diagram for VectSharp.Markdown.UnderlineFragment:



Public Member Functions

- UnderlineFragment (Point start, Point end, Colour colour, double thickness, string tag)

  Create a new UnderlineFragment.
- override void Translate (double deltaX, double deltaY)

  Applies the specified translation to the line fragment.

Parameters

deltaX	*The translation in the horizontal direction.*
deltaY	*The translation in the vertical direction.*

- override double GetAscent (double lineAscent)

  Get the ascent of the LineFragment.

Parameters

lineAscent	*The ascent of the Line on which the fragment should be drawn.*

Returns

The ascent of the LineFragment.

- override double GetMaxY ()

  Get the vertical coordinate of the bottom of the LineFragment.

  Returns

  The vertical coordinates of the bottom of the LineFragment.
- override double GetMaxX ()

  Get the horizontal coordinate of the right side of the LineFragment.

  Returns

  The horizontal coordinate of the right side of the LineFragment.
- override? Point Render (Graphics graphics, double deltaY)

  Render the LineFragment on the specified graphics surface.

Parameters

graphics	*The Graphics surface on which the LineFragment will be drawn.*
deltaY	*Additional translation on the vertical axis.*

Returns

If a text element has been drawn, this method should return the bottom-right corner of the drawn text element, otherwise `null`. This is used to update the MarkdownContext.BottomRight property.

Properties

- Point Start `[get]`

  The start position of the underline, relative to the cursor at the start of the line.
- Point End `[get]`

  The end position of the underline, relative to the cursor at the start of the line.
- Colour Colour `[get]`

  The colour of the underline.
- double Thickness `[get]`

  The thickness of the underline.

7.184.1 Detailed Description

Represents a text underline.

Definition at line 147 of file Line.cs.

7.184.2 Constructor & Destructor Documentation

7.184.2.1 UnderlineFragment()

```
VectSharp.Markdown.UnderlineFragment.UnderlineFragment (
            Point start,
            Point end,
            Colour colour,
            double thickness,
            string tag )
```

Create a new UnderlineFragment.

Parameters

start	The start position of the underline.
end	The end position of the underline.
colour	The colour of the underline.
thickness	The thickness of the underline.
tag	Graphics action tag for the underline.

Definition at line 177 of file Line.cs.

7.184.3 Member Function Documentation

7.184.3.1 GetAscent()

```
override double VectSharp.Markdown.UnderlineFragment.GetAscent (
            double lineAscent )  [virtual]
```

Get the ascent of the LineFragment.

Parameters

lineAscent	The ascent of the Line on which the fragment should be drawn.

Returns

The ascent of the LineFragment.

Implements VectSharp.Markdown.LineFragment.

Definition at line 194 of file Line.cs.

7.184.3.2 GetMaxX()

```
override double VectSharp.Markdown.UnderlineFragment.GetMaxX ( )  [virtual]
```

Get the horizontal coordinate of the right side of the LineFragment.

Returns

The horizontal coordinate of the right side of the LineFragment.

Implements VectSharp.Markdown.LineFragment.

Definition at line 206 of file Line.cs.

7.184.3.3 GetMaxY()

```
override double VectSharp.Markdown.UnderlineFragment.GetMaxY ( )  [virtual]
```

Get the vertical coordinate of the bottom of the LineFragment.

Returns

The vertical coordinates of the bottom of the LineFragment.

Implements VectSharp.Markdown.LineFragment.

Definition at line 200 of file Line.cs.

7.184.3.4 Render()

```
override?  Point VectSharp.Markdown.UnderlineFragment.Render (
            Graphics graphics,
            double deltaY )  [virtual]
```

Render the LineFragment on the specified *graphics* surface.

Parameters

graphics	The Graphics surface on which the LineFragment will be drawn.
deltaY	Additional translation on the vertical axis.

Returns

If a text element has been drawn, this method should return the bottom-right corner of the drawn text element, otherwise `null`. This is used to update the MarkdownContext.BottomRight property.

Implements VectSharp.Markdown.LineFragment.

Definition at line 212 of file Line.cs.

7.184.3.5 Translate()

```
override void VectSharp.Markdown.UnderlineFragment.Translate (
            double deltaX,
            double deltaY )  [virtual]
```

Applies the specified translation to the line fragment.

Parameters

deltaX	The translation in the horizontal direction.
deltaY	The translation in the vertical direction.

Implements VectSharp.Markdown.LineFragment.

Definition at line 187 of file Line.cs.

7.184.4 Property Documentation

7.184.4.1 Colour

```
Colour VectSharp.Markdown.UnderlineFragment.Colour  [get]
```

The colour of the underline.

Definition at line 162 of file Line.cs.

7.184.4.2 End

```
Point VectSharp.Markdown.UnderlineFragment.End  [get]
```

The end position of the underline, relative to the cursor at the start of the line.

Definition at line 157 of file Line.cs.

7.184.4.3 Start

```
Point VectSharp.Markdown.UnderlineFragment.Start  [get]
```

The start position of the underline, relative to the cursor at the start of the line.

Definition at line 152 of file Line.cs.

7.184.4.4 Thickness

```
double VectSharp.Markdown.UnderlineFragment.Thickness  [get]
```

The thickness of the underline.

Definition at line 167 of file Line.cs.

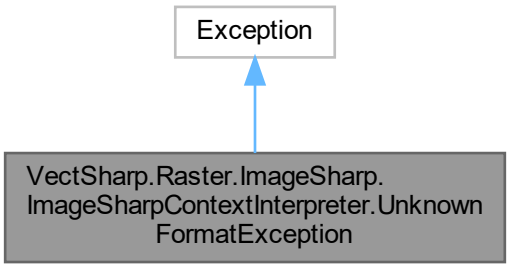
The documentation for this class was generated from the following file:

- VectSharp.Markdown/Line.cs

7.185 VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.UnknownFormatException Class Reference

The exception that is raised when the output file format is not specified and the file name does not have an extension corresponding to a known file format.

Inheritance diagram for VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.UnknownFormatException:



Properties

- string Format `[get]`

  The extension of the file that does not correspond to any known file format.

7.185.1 Detailed Description

The exception that is raised when the output file format is not specified and the file name does not have an extension corresponding to a known file format.

Definition at line 1110 of file ImageSharpContext.cs.

7.185.2 Property Documentation

7.185.2.1 Format

```
string VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.UnknownFormatException.Format
[get]
```

The extension of the file that does not correspond to any known file format.

Definition at line 1115 of file ImageSharpContext.cs.

The documentation for this class was generated from the following file:

- VectSharp.Raster.ImageSharp/ImageSharpContext.cs

7.186 VectSharp.TrueTypeFile.VerticalMetrics Struct Reference

Represents the maximum heigth above and depth below the baseline of a glyph.

Public Attributes

- int YMin

  The maximum depth below the baseline of the glyph.
- int YMax

  The maximum height above the baseline of the glyph.

7.186.1 Detailed Description

Represents the maximum heigth above and depth below the baseline of a glyph.

Definition at line 2376 of file TrueType.cs.

7.186.2 Member Data Documentation

7.186.2.1 YMax

```
int VectSharp.TrueTypeFile.VerticalMetrics.YMax
```

The maximum height above the baseline of the glyph.

Definition at line 2386 of file TrueType.cs.

7.186.2.2 YMin

```
int VectSharp.TrueTypeFile.VerticalMetrics.YMin
```

The maximum depth below the baseline of the glyph.

Definition at line 2381 of file TrueType.cs.

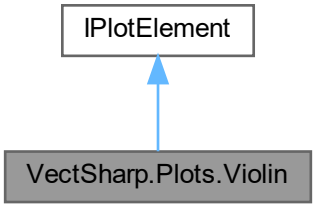
The documentation for this struct was generated from the following file:

- VectSharp/TrueType.cs

7.187 VectSharp.Plots.Violin Class Reference

A plot element that draws a violin plot.

Inheritance diagram for VectSharp.Plots.Violin:



Public Types

- enum ViolinSide

  The sides on which a violin can be drawn.

Public Member Functions

- Violin (IReadOnlyList< double > position, IReadOnlyList< double > direction, IReadOnlyList< double > data, ICoordinateSystem< IReadOnlyList< double > > coordinateSystem)

  Create a new Violin instance.
- void Plot (Graphics target)

  Draw the plot element on the specified target Graphics.

Parameters

target	*The Graphics on which to draw.*

Properties

- IReadOnlyList< double > Position `[get, set]`

  The position of the origin of the violin (e.g., the 0 in data space coordinates).
- IReadOnlyList< double > Direction `[get, set]`

  The direction along which the violin is drawn, in data space coordinates.
- double Width = 10 `[get, set]`

  The width of the violin in data space coordinates.
- bool Smooth = true `[get, set]`

  Determines whether the violin is smoothed or not.
- ViolinSide Sides = ViolinSide.Both `[get, set]`

  Determines on which side(s) the violin is drawn.
- IReadOnlyList< double > Data `[get, set]`

  The values whose distribution is displayed by the violin plot.
- int MaxBins = int.MaxValue `[get, set]`

  Maximum number of bins.
- PlotElementPresentationAttributes PresentationAttributes = new PlotElementPresentationAttributes() { Fill = Colours.White } `[get, set]`

  Presentation attributes for the violin plot.
- ICoordinateSystem< IReadOnlyList< double > > CoordinateSystem `[get, set]`

  The coordinate system used to transform the points from data space to plot space.
- string Tag `[get, set]`

  A tag to identify the violin in the plot.

7.187.1 Detailed Description

A plot element that draws a violin plot.

Definition at line 27 of file Violin.cs.

7.187.2 Member Enumeration Documentation

7.187.2.1 ViolinSide

`enum VectSharp.Plots.Violin.ViolinSide`

The sides on which a violin can be drawn.

Definition at line 32 of file Violin.cs.

7.187.3 Constructor & Destructor Documentation

7.187.3.1 Violin()

```
VectSharp.Plots.Violin.Violin (
            IReadOnlyList< double > position,
            IReadOnlyList< double > direction,
            IReadOnlyList< double > data,
            ICoordinateSystem< IReadOnlyList< double > > coordinateSystem )
```

Create a new Violin instance.

Parameters

position	The position of the origin of the violin (e.g., the 0 in data space coordinates).
direction	The direction along which the violin is drawn, in data space coordinates.
data	The values whose distribution is displayed by the violin plot.
coordinateSystem	The coordinate system used to transform the points from data space to plot space.

Definition at line 108 of file Violin.cs.

7.187.4 Member Function Documentation

7.187.4.1 Plot()

```
void VectSharp.Plots.Violin.Plot (
            Graphics target )
```

Draw the plot element on the specified *target* Graphics.

Parameters

target	The Graphics on which to draw.

Implements VectSharp.Plots.IPlotElement.

Definition at line 117 of file Violin.cs.

7.187.5 Property Documentation

7.187.5.1 CoordinateSystem

```
ICoordinateSystem<IReadOnlyList<double> > VectSharp.Plots.Violin.CoordinateSystem  [get],
[set]
```

The coordinate system used to transform the points from data space to plot space.

Implements VectSharp.Plots.IPlotElement.

Definition at line 93 of file Violin.cs.

7.187.5.2 Data

```
IReadOnlyList<double> VectSharp.Plots.Violin.Data  [get], [set]
```

The values whose distribution is displayed by the violin plot.

Definition at line 78 of file Violin.cs.

7.187.5.3 Direction

```
IReadOnlyList<double> VectSharp.Plots.Violin.Direction  [get], [set]
```

The direction along which the violin is drawn, in data space coordinates.

Definition at line 58 of file Violin.cs.

7.187.5.4 MaxBins

```
int VectSharp.Plots.Violin.MaxBins = int.MaxValue  [get], [set]
```

Maximum number of bins.

Definition at line 83 of file Violin.cs.

7.187.5.5 Position

```
IReadOnlyList<double> VectSharp.Plots.Violin.Position  [get], [set]
```

The position of the origin of the violin (e.g., the 0 in data space coordinates).

Definition at line 53 of file Violin.cs.

7.187.5.6 PresentationAttributes

```
PlotElementPresentationAttributes VectSharp.Plots.Violin.PresentationAttributes = new PlotElementPresentationA
{ Fill = Colours.White }  [get], [set]
```

Presentation attributes for the violin plot.

Definition at line 88 of file Violin.cs.

7.187.5.7 Sides

```
ViolinSide VectSharp.Plots.Violin.Sides = ViolinSide.Both  [get], [set]
```

Determines on which side(s) the violin is drawn.

Definition at line 73 of file Violin.cs.

7.187.5.8 Smooth

```
bool VectSharp.Plots.Violin.Smooth = true  [get], [set]
```

Determines whether the violin is smoothed or not.

Definition at line 68 of file Violin.cs.

7.187.5.9 Tag

```
string VectSharp.Plots.Violin.Tag  [get], [set]
```

A tag to identify the violin in the plot.

Definition at line 99 of file Violin.cs.

7.187.5.10 Width

```
double VectSharp.Plots.Violin.Width = 10  [get], [set]
```

The width of the violin in data space coordinates.

Definition at line 63 of file Violin.cs.

The documentation for this class was generated from the following file:

- VectSharp.Plots/Violin.cs

Chapter 8

File Documentation

8.1 AnimatedCanvas.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2022-2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using Avalonia;
00019 using Avalonia.Media;
00020 using System;
00021 using System.Collections.Generic;
00022
00023 namespace VectSharp.Canvas
00024 {
00025 /// <summary>
00026 /// An <see cref="Avalonia.Controls.Canvas"/> containing an animation.
00027 /// </summary>
00028     public class AnimatedCanvas :  Avalonia.Controls.Control
00029     {
00030 /// <summary>
00031 /// Defines the <see cref="CurrentFrame"/> property.
00032 /// </summary>
00033         public static readonly StyledProperty<int> CurrentFrameProperty =
     AvaloniaProperty.Register<AnimatedCanvas, int>(nameof(CurrentFrame), coerce:  (ownerObject, val) =>
00034         {
00035             AnimatedCanvas owner = (AnimatedCanvas)ownerObject;
00036
00037             if (owner.maxFrameIndex > 0)
00038             {
00039                 if (val < owner.maxFrameIndex)
00040                 {
00041                     return val;
00042                 }
00043                 else
00044                 {
00045                     owner.IsPlaying = false;
00046                     return owner.maxFrameIndex - 1;
00047                 }
00048             }
00049             else
00050             {
00051                 return val % owner.RenderActions.Length;
00052             }
00053         });
00054
00055 /// <summary>
00056 /// The current frame in the animation.
00057 /// </summary>
```

```
00058         public int CurrentFrame
00059         {
00060             get { return GetValue(CurrentFrameProperty); }
00061             set { SetValue(CurrentFrameProperty, value); }
00062         }
00063
00064 /// <summary>
00065 /// Defines the <see cref="FrameCount"/> property.
00066 /// </summary>
00067         public static readonly DirectProperty<AnimatedCanvas, int> FrameCountProperty =
      AvaloniaProperty.RegisterDirect<AnimatedCanvas, int>(nameof(FrameCount), o => o.maxFrameIndex);
00068
00069         private int frameCount;
00070
00071 /// <summary>
00072 /// The number of frames in the animation.
00073 /// </summary>
00074         public int FrameCount
00075         {
00076             get
00077             {
00078                 return frameCount;
00079             }
00080             private set
00081             {
00082                 SetAndRaise(FrameCountProperty, ref frameCount, value);
00083             }
00084         }
00085
00086         private int maxFrameIndex { get; }
00087
00088 /// <summary>
00089 /// Defines the <see cref="FrameRate"/> property.
00090 /// </summary>
00091         public static readonly DirectProperty<AnimatedCanvas, double> FrameRateProperty =
      AvaloniaProperty.RegisterDirect<AnimatedCanvas, double>(nameof(FrameRate), o => o.FrameRate);
00092
00093         private double frameRate;
00094
00095 /// <summary>
00096 /// The target frame rate of the animation.
00097 /// </summary>
00098         public double FrameRate
00099         {
00100             get
00101             {
00102                 return frameRate;
00103             }
00104             private set
00105             {
00106                 SetAndRaise(FrameRateProperty, ref frameRate, value);
00107             }
00108         }
00109
00110
00111
00112 /// <summary>
00113 /// Defines the <see cref="IsPlaying"/> property.
00114 /// </summary>
00115         public static readonly StyledProperty<bool> IsPlayingProperty =
      AvaloniaProperty.Register<AnimatedCanvas, bool>(nameof(IsPlaying), false, coerce:  (ownerObject, val)
      =>
00116         {
00117             AnimatedCanvas owner = (AnimatedCanvas)ownerObject;
00118
00119             if (owner.IsPlaying && !val)
00120             {
00121                 owner.RefreshTimer.Dispose();
00122             }
00123             else if (!owner.IsPlaying && val)
00124             {
00125                 owner.RefreshTimer = new System.Threading.Timer(_ =>
00126                 {
00127                     _ = Avalonia.Threading.Dispatcher.UIThread.InvokeAsync(() =>
00128                     {
00129                         owner.CurrentFrame++;
00130                     });
00131                 }, null, 0, (long)(1000.0 / owner.FrameRate));
00132             }
00133
00134             return val;
00135         });
00136
00137 /// <summary>
00138 /// The current frame in the animation.
00139 /// </summary>
00140         public bool IsPlaying
```

```
00141         {
00142             get { return GetValue(IsPlayingProperty); }
00143             set { SetValue(IsPlayingProperty, value); }
00144         }
00145 
00146 
00147         private List<RenderAction>[] RenderActions;
00148 
00149         static Avalonia.Point Origin = new Avalonia.Point(0, 0);
00150 
00151         private SolidColorBrush BackgroundBrush;
00152 
00153         private Dictionary<string, (IImage, bool)> Images;
00154 
00155         private System.Threading.Timer RefreshTimer { get; set; }
00156 
00157         static AnimatedCanvas()
00158         {
00159             AffectsRender<AnimatedCanvas>(CurrentFrameProperty);
00160         }
00161 
00162         internal AnimatedCanvas(VectSharp.Animation animation, double durationScaling, double
      frameRate, AvaloniaContextInterpreter.TextOptions textOption, FilterOption filterOption)
00163         {
00164             Colour backgroundColour = animation.Background;
00165 
00166             this.BackgroundBrush = new SolidColorBrush(Color.FromArgb((byte)(backgroundColour.A *
      255), (byte)(backgroundColour.R * 255), (byte)(backgroundColour.G * 255), (byte)(backgroundColour.B *
      255)));
00167 
00168             this.Width = animation.Width;
00169             this.Height = animation.Height;
00170             this.Images = new Dictionary<string, (IImage, bool)>();
00171 
00172             int frames = (int)Math.Ceiling(animation.Duration * frameRate * durationScaling / 1000);
00173 
00174             this.RenderActions = new List<RenderAction>[frames];
00175 
00176             for (int i = 0; i < frames; i++)
00177             {
00178                 double frameTime = i / frameRate / durationScaling * 1000;
00179 
00180                 Page pag = animation.GetFrameAtAbsolute(frameTime);
00181 
00182                 AvaloniaDrawingContext ctx = new AvaloniaDrawingContext(this.Width, this.Height,
      false, textOption, this.Images, filterOption);
00183 
00184                 pag.Graphics.CopyToIGraphicsContext(ctx);
00185                 this.RenderActions[i] = ctx.RenderActions;
00186             }
00187 
00188             if (animation.RepeatCount > 0)
00189             {
00190                 this.maxFrameIndex = animation.RepeatCount * frames;
00191             }
00192             else
00193             {
00194                 this.maxFrameIndex = 0;
00195             }
00196 
00197             this.FrameCount = frames;
00198 
00199             this.FrameRate = frameRate;
00200         }
00201 
00202 /// <inheritdoc/>
00203         public override void Render(DrawingContext context)
00204         {
00205             context.FillRectangle(this.BackgroundBrush, new Avalonia.Rect(0, 0, Width, Height));
00206 
00207             foreach (RenderAction act in this.RenderActions[this.CurrentFrame %
      this.RenderActions.Length])
00208             {
00209                 if (act.ActionType == RenderAction.ActionTypes.Path)
00210                 {
00211                     DrawingContext.PushedState?  state = null;
00212 
00213                     if (act.ClippingPath != null)
00214                     {
00215                         //Random draw operation needed due to
      https://github.com/AvaloniaUI/Avalonia/issues/4408
00216                         context.DrawGeometry(null, new Pen(Brushes.Transparent), act.ClippingPath);
00217                         state = context.PushGeometryClip(act.ClippingPath);
00218                     }
00219 
00220                     using (context.PushTransform(act.Transform))
00221                     {
```

```
00222                         context.DrawGeometry(act.Fill, act.Stroke, act.Geometry);
00223                     }
00224 
00225                     if (state != null)
00226                     {
00227                         state?.Dispose();
00228                         //Random draw operation needed due to
      https://github.com/AvaloniaUI/Avalonia/issues/4408
00229                         context.DrawGeometry(null, new Pen(Brushes.Transparent), act.ClippingPath);
00230                     }
00231                 }
00232                 else if (act.ActionType == RenderAction.ActionTypes.Text)
00233                 {
00234                     DrawingContext.PushedState?  state = null;
00235 
00236                     if (act.ClippingPath != null)
00237                     {
00238                         //Random draw operation needed due to
      https://github.com/AvaloniaUI/Avalonia/issues/4408
00239                         context.DrawGeometry(null, new Pen(Brushes.Transparent), act.ClippingPath);
00240                         state = context.PushGeometryClip(act.ClippingPath);
00241                     }
00242 
00243                     using (context.PushTransform(act.Transform))
00244                     {
00245                         context.DrawText(act.Text, Origin);
00246                     }
00247 
00248                     if (state != null)
00249                     {
00250                         state?.Dispose();
00251                         //Random draw operation needed due to
      https://github.com/AvaloniaUI/Avalonia/issues/4408
00252                         context.DrawGeometry(null, new Pen(Brushes.Transparent), act.ClippingPath);
00253                     }
00254                 }
00255                 else if (act.ActionType == RenderAction.ActionTypes.RasterImage)
00256                 {
00257                     DrawingContext.PushedState?  state = null;
00258 
00259                     if (act.ClippingPath != null)
00260                     {
00261                         //Random draw operation needed due to
      https://github.com/AvaloniaUI/Avalonia/issues/4408
00262                         context.DrawGeometry(null, new Pen(Brushes.Transparent), act.ClippingPath);
00263                         state = context.PushGeometryClip(act.ClippingPath);
00264                     }
00265 
00266                     (IImage, bool) image = Images[act.ImageId];
00267 
00268                     using (context.PushTransform(act.Transform))
00269                     {
00270                         RenderOptions.SetBitmapInterpolationMode(this, image.Item2 ?
      Avalonia.Media.Imaging.BitmapInterpolationMode.HighQuality :
      Avalonia.Media.Imaging.BitmapInterpolationMode.None);
00271                         context.DrawImage(image.Item1, act.ImageSource.Value,
      act.ImageDestination.Value);
00272                     }
00273 
00274                     if (state != null)
00275                     {
00276                         state?.Dispose();
00277                         //Random draw operation needed due to
      https://github.com/AvaloniaUI/Avalonia/issues/4408
00278                         context.DrawGeometry(null, new Pen(Brushes.Transparent), act.ClippingPath);
00279                     }
00280                 }
00281             }
00282 
00283         }
00284 
00285 /// <inheritdoc/>
00286         protected override void OnDetachedFromVisualTree(VisualTreeAttachmentEventArgs e)
00287         {
00288             base.OnDetachedFromVisualTree(e);
00289             IsPlaying = false;
00290         }
00291 
00292 /// <inheritdoc/>
00293         protected override void OnAttachedToVisualTree(VisualTreeAttachmentEventArgs e)
00294         {
00295             base.OnAttachedToVisualTree(e);
00296             IsPlaying = true;
00297         }
00298     }
00299 
00300 }
```

8.2 AvaloniaContext.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using Avalonia.Controls;
00019 using Avalonia.Controls.Shapes;
00020 using Avalonia.Media;
00021 using Avalonia.Media.Imaging;
00022 using Avalonia.Media.TextFormatting;
00023 using System;
00024 using System.Collections.Generic;
00025 using System.Linq;
00026 using System.Runtime.InteropServices;
00027 using VectSharp.Filters;
00028
00029 namespace VectSharp.Canvas
00030 {
00031     // Adapted from
      https://github.com/AvaloniaUI/Avalonia/blob/412438d334f970367834d52b92b9ed3e89f3d18c/src/Avalonia.Controls/TextBlock.cs#
00032     internal readonly struct SimpleTextSource :  ITextSource
00033     {
00034         private readonly string _text;
00035         private readonly TextRunProperties _defaultProperties;
00036
00037         public SimpleTextSource(string text, TextRunProperties defaultProperties)
00038         {
00039             _text = text;
00040             _defaultProperties = defaultProperties;
00041         }
00042
00043         public TextRun GetTextRun(int textSourceIndex)
00044         {
00045             if (textSourceIndex > _text.Length)
00046             {
00047                 return new TextEndOfParagraph();
00048             }
00049
00050             var runText = _text.AsMemory(textSourceIndex);
00051
00052             if (runText.IsEmpty)
00053             {
00054                 return new TextEndOfParagraph();
00055             }
00056
00057             return new TextCharacters(runText, _defaultProperties);
00058         }
00059     }
00060
00061     internal static class MatrixUtils
00062     {
00063         public static Avalonia.Matrix ToAvaloniaMatrix(this double[,] matrix)
00064         {
00065             return new Avalonia.Matrix(matrix[0, 0], matrix[1, 0], matrix[0, 1], matrix[1, 1],
      matrix[0, 2], matrix[1, 2]);
00066         }
00067
00068         public static double[] Multiply(double[,] matrix, double[] vector)
00069         {
00070             double[] tbr = new double[2];
00071
00072             tbr[0] = matrix[0, 0] * vector[0] + matrix[0, 1] * vector[1] + matrix[0, 2];
00073             tbr[1] = matrix[1, 0] * vector[0] + matrix[1, 1] * vector[1] + matrix[1, 2];
00074
00075             return tbr;
00076         }
00077
00078         public static double[,] Multiply(double[,] matrix1, double[,] matrix2)
00079         {
00080             double[,] tbr = new double[3, 3];
00081
00082             for (int i = 0; i < 3; i++)
00083             {
```

```
00084                 for (int j = 0; j < 3; j++)
00085                 {
00086                     for (int k = 0; k < 3; k++)
00087                     {
00088                         tbr[i, j] += matrix1[i, k] * matrix2[k, j];
00089                     }
00090                 }
00091             }
00092 
00093             return tbr;
00094         }
00095 
00096         public static double[,] Rotate(double[,] matrix, double angle)
00097         {
00098             double[,] rotationMatrix = new double[3, 3];
00099             rotationMatrix[0, 0] = Math.Cos(angle);
00100             rotationMatrix[0, 1] = -Math.Sin(angle);
00101             rotationMatrix[1, 0] = Math.Sin(angle);
00102             rotationMatrix[1, 1] = Math.Cos(angle);
00103             rotationMatrix[2, 2] = 1;
00104 
00105             return Multiply(matrix, rotationMatrix);
00106         }
00107 
00108         public static double[,] Translate(double[,] matrix, double x, double y)
00109         {
00110             double[,] translationMatrix = new double[3, 3];
00111             translationMatrix[0, 0] = 1;
00112             translationMatrix[0, 2] = x;
00113             translationMatrix[1, 1] = 1;
00114             translationMatrix[1, 2] = y;
00115             translationMatrix[2, 2] = 1;
00116 
00117             return Multiply(matrix, translationMatrix);
00118         }
00119 
00120         public static double[,] Scale(double[,] matrix, double scaleX, double scaleY)
00121         {
00122             double[,] scaleMatrix = new double[3, 3];
00123             scaleMatrix[0, 0] = scaleX;
00124             scaleMatrix[1, 1] = scaleY;
00125             scaleMatrix[2, 2] = 1;
00126 
00127             return Multiply(matrix, scaleMatrix);
00128         }
00129 
00130         public static double[,] Identity = new double[,] { { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0, 1 } };
00131         public static double[,] Invert(double[,] m)
00132         {
00133             double[,] tbr = new double[3, 3];
00134 
00135             tbr[0, 0] = (m[1, 1] * m[2, 2] - m[1, 2] * m[2, 1]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00136             tbr[0, 1] = -(m[0, 1] * m[2, 2] - m[0, 2] * m[2, 1]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00137             tbr[0, 2] = (m[0, 1] * m[1, 2] - m[0, 2] * m[1, 1]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00138             tbr[1, 0] = -(m[1, 0] * m[2, 2] - m[1, 2] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00139             tbr[1, 1] = (m[0, 0] * m[2, 2] - m[0, 2] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00140             tbr[1, 2] = -(m[0, 0] * m[1, 2] - m[0, 2] * m[1, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00141             tbr[2, 0] = (m[1, 0] * m[2, 1] - m[1, 1] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00142             tbr[2, 1] = -(m[0, 0] * m[2, 1] - m[0, 1] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00143             tbr[2, 2] = (m[0, 0] * m[1, 1] - m[0, 1] * m[1, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00144 
00145             return tbr;
00146         }
00147     }
00148 
00149     internal static class Utils
00150     {
00151         public static void CoerceNaNAndInfinityToZero(ref double val)
00152         {
```

```
00153             if (double.IsNaN(val) || double.IsInfinity(val) || val == double.MinValue || val ==
      double.MaxValue)
00154             {
00155                 val = 0;
00156             }
00157         }
00158 
00159         public static void CoerceNaNAndInfinityToZero(ref double val1, ref double val2)
00160         {
00161             if (double.IsNaN(val1) || double.IsInfinity(val1) || val1 == double.MinValue || val1 ==
      double.MaxValue)
00162             {
00163                 val1 = 0;
00164             }
00165 
00166             if (double.IsNaN(val2) || double.IsInfinity(val2) || val2 == double.MinValue || val2 ==
      double.MaxValue)
00167             {
00168                 val2 = 0;
00169             }
00170         }
00171 
00172         public static void CoerceNaNAndInfinityToZero(ref double val1, ref double val2, ref double
      val3, ref double val4)
00173         {
00174             if (double.IsNaN(val1) || double.IsInfinity(val1) || val1 == double.MinValue || val1 ==
      double.MaxValue)
00175             {
00176                 val1 = 0;
00177             }
00178 
00179             if (double.IsNaN(val2) || double.IsInfinity(val2) || val2 == double.MinValue || val2 ==
      double.MaxValue)
00180             {
00181                 val2 = 0;
00182             }
00183 
00184             if (double.IsNaN(val3) || double.IsInfinity(val3) || val3 == double.MinValue || val3 ==
      double.MaxValue)
00185             {
00186                 val3 = 0;
00187             }
00188 
00189             if (double.IsNaN(val4) || double.IsInfinity(val4) || val4 == double.MinValue || val4 ==
      double.MaxValue)
00190             {
00191                 val4 = 0;
00192             }
00193         }
00194 
00195         public static void CoerceNaNAndInfinityToZero(ref double val1, ref double val2, ref double
      val3, ref double val4, ref double val5, ref double val6)
00196         {
00197             if (double.IsNaN(val1) || double.IsInfinity(val1) || val1 == double.MinValue || val1 ==
      double.MaxValue)
00198             {
00199                 val1 = 0;
00200             }
00201 
00202             if (double.IsNaN(val2) || double.IsInfinity(val2) || val2 == double.MinValue || val2 ==
      double.MaxValue)
00203             {
00204                 val2 = 0;
00205             }
00206 
00207             if (double.IsNaN(val3) || double.IsInfinity(val3) || val3 == double.MinValue || val3 ==
      double.MaxValue)
00208             {
00209                 val3 = 0;
00210             }
00211 
00212             if (double.IsNaN(val4) || double.IsInfinity(val4) || val4 == double.MinValue || val4 ==
      double.MaxValue)
00213             {
00214                 val4 = 0;
00215             }
00216 
00217             if (double.IsNaN(val5) || double.IsInfinity(val5) || val5 == double.MinValue || val5 ==
      double.MaxValue)
00218             {
00219                 val5 = 0;
00220             }
00221 
00222             if (double.IsNaN(val6) || double.IsInfinity(val6) || val6 == double.MinValue || val6 ==
      double.MaxValue)
00223             {
00224                 val6 = 0;
```

```
00225             }
00226         }
00227     }
00228
00229     internal class AvaloniaContext :  IGraphicsContext
00230     {
00231         public Dictionary<string, Delegate> TaggedActions { get; set; } = new Dictionary<string,
      Delegate>();
00232
00233         private bool removeTaggedActions = true;
00234
00235         public string Tag { get; set; }
00236
00237         private AvaloniaContextInterpreter.TextOptions _textOption;
00238
00239         private Avalonia.Controls.Canvas currControlElement;
00240
00241         private Stack<Avalonia.Controls.Canvas> controlElements;
00242         private FilterOption _filterOption;
00243
00244         public AvaloniaContext(double width, double height, bool removeTaggedActionsAfterExecution,
      AvaloniaContextInterpreter.TextOptions textOption, FilterOption filterOption)
00245         {
00246             currentPath = new PathGeometry();
00247             currentFigure = new PathFigure() { IsClosed = false };
00248             figureInitialised = false;
00249             ControlItem = new Avalonia.Controls.Canvas() { Width = width, Height = height,
      ClipToBounds = true };
00250             removeTaggedActions = removeTaggedActionsAfterExecution;
00251
00252             _transform = new double[3, 3];
00253
00254             _transform[0, 0] = 1;
00255             _transform[1, 1] = 1;
00256             _transform[2, 2] = 1;
00257
00258             states = new Stack<double[,]>();
00259
00260             _textOption = textOption;
00261
00262             currControlElement = ControlItem;
00263             controlElements = new Stack<Avalonia.Controls.Canvas>();
00264             controlElements.Push(ControlItem);
00265             _filterOption = filterOption;
00266         }
00267
00268         public Avalonia.Controls.Canvas ControlItem { get; }
00269
00270         public double Width { get { return ControlItem.Width; } }
00271         public double Height { get { return ControlItem.Height; } }
00272
00273         public void Translate(double x, double y)
00274         {
00275             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00276
00277             _transform = MatrixUtils.Translate(_transform, x, y);
00278
00279             currentPath = new PathGeometry();
00280             currentFigure = new PathFigure() { IsClosed = false };
00281             figureInitialised = false;
00282         }
00283
00284         public TextBaselines TextBaseline { get; set; }
00285
00286         private void PathText(string text, double x, double y)
00287         {
00288             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00289
00290             GraphicsPath textPath = new GraphicsPath().AddText(x, y, text, Font, TextBaseline);
00291
00292             for (int j = 0; j < textPath.Segments.Count; j++)
00293             {
00294                 switch (textPath.Segments[j].Type)
00295                 {
00296                     case VectSharp.SegmentType.Move:
00297                         this.MoveTo(textPath.Segments[j].Point.X, textPath.Segments[j].Point.Y);
00298                         break;
00299                     case VectSharp.SegmentType.Line:
00300                         this.LineTo(textPath.Segments[j].Point.X, textPath.Segments[j].Point.Y);
00301                         break;
00302                     case VectSharp.SegmentType.CubicBezier:
00303                         this.CubicBezierTo(textPath.Segments[j].Points[0].X,
      textPath.Segments[j].Points[0].Y, textPath.Segments[j].Points[1].X, textPath.Segments[j].Points[1].Y,
      textPath.Segments[j].Points[2].X, textPath.Segments[j].Points[2].Y);
00304                         break;
00305                     case VectSharp.SegmentType.Close:
00306                         this.Close();
```

```
00307                       break;
00308                   }
00309               }
00310           }
00311
00312           public void StrokeText(string text, double x, double y)
00313           {
00314               Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00315
00316               PathText(text, x, y);
00317               Stroke();
00318           }
00319
00320           public void FillText(string text, double x, double y)
00321           {
00322               Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00323
00324               if (_textOption == AvaloniaContextInterpreter.TextOptions.NeverConvert || (_textOption ==
      AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary && Font.FontFamily.IsStandardFamily &&
      Font.FontFamily.FileName != "ZapfDingbats" && Font.FontFamily.FileName != "Symbol"))
00325               {
00326
00327                   TextBlock blk = new TextBlock() { ClipToBounds = false, Text = text, FontFamily =
      Avalonia.Media.FontFamily.Parse(FontFamily), FontSize = Font.FontSize, FontStyle =
      (Font.FontFamily.IsOblique ?  FontStyle.Oblique :  Font.FontFamily.IsItalic ?  FontStyle.Italic :
      FontStyle.Normal), FontWeight = (Font.FontFamily.IsBold ?  FontWeight.Bold :  FontWeight.Regular),
      TextAlignment = TextAlignment.Left };
00328
00329                   double top = y;
00330                   double left = x;
00331
00332                   double[,] currTransform = null;
00333                   double[,] deltaTransform = MatrixUtils.Identity;
00334
00335                   if (Font.FontFamily.TrueTypeFile != null)
00336                   {
00337                       currTransform = MatrixUtils.Translate(_transform, x, y);
00338                   }
00339
00340                   Font.DetailedFontMetrics metrics = Font.MeasureTextAdvanced(text);
00341
00342                   if (text.StartsWith(" "))
00343                   {
00344                       var spaceMetrics = Font.MeasureTextAdvanced(" ");
00345
00346                       for (int i = 0; i < text.Length; i++)
00347                       {
00348                           if (text[i] == ' ')
00349                           {
00350                               left -= spaceMetrics.AdvanceWidth;
00351                           }
00352                           else
00353                           {
00354                               break;
00355                           }
00356                       }
00357                   }
00358
00359                   if (TextBaseline == TextBaselines.Top)
00360                   {
00361                       blk.VerticalAlignment = Avalonia.Layout.VerticalAlignment.Top;
00362
00363                       if (Font.FontFamily.TrueTypeFile != null)
00364                       {
00365                           if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
00366                           {
00367                               currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing * 2, top + metrics.Top - Font.WinAscent);
00368                               deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing * 2, top + metrics.Top - Font.WinAscent);
00369                           }
00370                           else
00371                           {
00372                               currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing * 2, top + metrics.Top - Font.Ascent);
00373                               deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing * 2, top + metrics.Top - Font.Ascent);
00374                           }
00375                       }
00376                   }
00377                   else if (TextBaseline == TextBaselines.Middle)
00378                   {
00379                       blk.VerticalAlignment = Avalonia.Layout.VerticalAlignment.Top;
00380
00381                       if (Font.FontFamily.TrueTypeFile != null)
00382                       {
00383                           if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
```

```
00384                         {
00385                             currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing * 2, top + metrics.Top / 2 + metrics.Bottom / 2 - Font.WinAscent);
00386                             deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing * 2, top + metrics.Top / 2 + metrics.Bottom / 2 - Font.WinAscent);
00387                         }
00388                         else
00389                         {
00390                             currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing * 2, top + metrics.Top / 2 + metrics.Bottom / 2 - Font.Ascent);
00391                             deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing * 2, top + metrics.Top / 2 + metrics.Bottom / 2 - Font.Ascent);
00392                         }
00393 
00394                     }
00395                 }
00396                 else if (TextBaseline == TextBaselines.Baseline)
00397                 {
00398                     blk.VerticalAlignment = Avalonia.Layout.VerticalAlignment.Top;
00399 
00400                     if (Font.FontFamily.TrueTypeFile != null)
00401                     {
00402                         if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
00403                         {
00404                             currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing * 2, top - Font.WinAscent);
00405                             deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing * 2, top - Font.YMax);
00406                         }
00407                         else
00408                         {
00409                             currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing * 2, top - Font.Ascent);
00410                             deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing * 2, top - Font.YMax);
00411                         }
00412                     }
00413                 }
00414                 else if (TextBaseline == TextBaselines.Bottom)
00415                 {
00416                     blk.VerticalAlignment = Avalonia.Layout.VerticalAlignment.Bottom;
00417 
00418                     if (Font.FontFamily.TrueTypeFile != null)
00419                     {
00420                         if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
00421                         {
00422                             currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing * 2, top - Font.WinAscent + metrics.Bottom);
00423                             deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing * 2, top - Font.WinAscent + metrics.Bottom);
00424                         }
00425                         else
00426                         {
00427                             currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing * 2, top - Font.Ascent + metrics.Bottom);
00428                             deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing * 2, top - Font.Ascent + metrics.Bottom);
00429                         }
00430                     }
00431                 }
00432 
00433                 blk.RenderTransform = new MatrixTransform(currTransform.ToAvaloniaMatrix());
00434                 blk.RenderTransformOrigin = new Avalonia.RelativePoint(0, 0,
      Avalonia.RelativeUnit.Absolute);
00435 
00436                 Avalonia.Media.Brush foreground = null;
00437 
00438                 if (this.FillStyle is SolidColourBrush solid)
00439                 {
00440                     foreground = new SolidColorBrush(Color.FromArgb(FillAlpha, (byte)(solid.R * 255),
      (byte)(solid.G * 255), (byte)(solid.B * 255)));
00441                 }
00442                 else if (this.FillStyle is LinearGradientBrush linearGradient)
00443                 {
00444                     foreground = linearGradient.ToLinearGradientBrush(deltaTransform);
00445                 }
00446                 else if (this.FillStyle is RadialGradientBrush radialGradient)
00447                 {
00448                     foreground = radialGradient.ToRadialGradientBrush(metrics.Width +
      metrics.LeftSideBearing + metrics.RightSideBearing, deltaTransform);
00449                 }
00450 
00451                 blk.Foreground = foreground;
00452 
00453                 currControlElement.Children.Add(blk);
00454 
00455                 if (!string.IsNullOrEmpty(Tag))
```

```
00456                 {
00457                     if (TaggedActions.ContainsKey(Tag))
00458                     {
00459                         TaggedActions[Tag].DynamicInvoke(blk);
00460 
00461                         if (removeTaggedActions)
00462                         {
00463                             TaggedActions.Remove(Tag);
00464                         }
00465                     }
00466                 }
00467             }
00468             else
00469             {
00470                 PathText(text, x, y);
00471                 Fill(FillRule.NonZeroWinding);
00472             }
00473         }
00474 
00475         public Brush StrokeStyle { get; private set; } = Colour.FromRgb(0, 0, 0);
00476         private byte StrokeAlpha = 255;
00477 
00478         public Brush FillStyle { get; private set; } = Colour.FromRgb(0, 0, 0);
00479         private byte FillAlpha = 255;
00480 
00481         public void SetFillStyle((int r, int g, int b, double a) style)
00482         {
00483             FillStyle = Colour.FromRgba(style.r, style.g, style.b, (int)(style.a * 255));
00484             FillAlpha = (byte)(style.a * 255);
00485         }
00486 
00487         public void SetFillStyle(Brush style)
00488         {
00489             FillStyle = style;
00490 
00491             if (style is SolidColourBrush solid)
00492             {
00493                 FillAlpha = (byte)(solid.A * 255);
00494             }
00495             else
00496             {
00497                 FillAlpha = 255;
00498             }
00499         }
00500 
00501         public void SetStrokeStyle((int r, int g, int b, double a) style)
00502         {
00503             StrokeStyle = Colour.FromRgba(style.r, style.g, style.b, (int)(style.a * 255));
00504             StrokeAlpha = (byte)(style.a * 255);
00505         }
00506 
00507         public void SetStrokeStyle(Brush style)
00508         {
00509             StrokeStyle = style;
00510 
00511             if (style is SolidColourBrush solid)
00512             {
00513                 StrokeAlpha = (byte)(solid.A * 255);
00514             }
00515             else
00516             {
00517                 StrokeAlpha = 255;
00518             }
00519         }
00520 
00521         private LineDash LineDash;
00522 
00523         public void SetLineDash(LineDash dash)
00524         {
00525             LineDash = new LineDash(dash.DashArray, dash.Phase);
00526         }
00527 
00528         public void Rotate(double angle)
00529         {
00530             Utils.CoerceNaNAndInfinityToZero(ref angle);
00531 
00532             _transform = MatrixUtils.Rotate(_transform, angle);
00533 
00534             currentPath = new PathGeometry();
00535             currentFigure = new PathFigure() { IsClosed = false };
00536             figureInitialised = false;
00537         }
00538 
00539         public void Transform(double a, double b, double c, double d, double e, double f)
00540         {
00541             Utils.CoerceNaNAndInfinityToZero(ref a, ref b, ref c, ref d, ref e, ref f);
00542 
```

```
00543             double[,] transfMatrix = new double[3, 3] { { a, c, e }, { b, d, f }, { 0, 0, 1 } };
00544             _transform = MatrixUtils.Multiply(_transform, transfMatrix);
00545 
00546             currentPath = new PathGeometry();
00547             currentFigure = new PathFigure() { IsClosed = false };
00548             figureInitialised = false;
00549         }
00550 
00551         public void Scale(double x, double y)
00552         {
00553             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00554 
00555             _transform = MatrixUtils.Scale(_transform, x, y);
00556 
00557             currentPath = new PathGeometry();
00558             currentFigure = new PathFigure() { IsClosed = false };
00559             figureInitialised = false;
00560         }
00561 
00562         private double[,] _transform;
00563 
00564         private readonly Stack<double[,]> states;
00565 
00566         public void Save()
00567         {
00568             states.Push((double[,])_transform.Clone());
00569             controlElements.Push(currControlElement);
00570         }
00571 
00572         public void Restore()
00573         {
00574             _transform = states.Pop();
00575             currControlElement = controlElements.Pop();
00576         }
00577 
00578         public double LineWidth { get; set; }
00579         public LineCaps LineCap { get; set; }
00580         public LineJoins LineJoin { get; set; }
00581 
00582         private string FontFamily;
00583         private Font _Font;
00584 
00585         public Font Font
00586         {
00587             get
00588             {
00589                 return _Font;
00590             }
00591 
00592             set
00593             {
00594                 _Font = value;
00595 
00596                 if (Font.FontFamily is ResourceFontFamily fam)
00597                 {
00598                     FontFamily = fam.ResourceName;
00599                 }
00600                 else
00601                 {
00602                     if (!Font.FontFamily.IsStandardFamily)
00603                     {
00604                         if (Font.FontFamily.TrueTypeFile != null)
00605                         {
00606                             FontFamily = Font.FontFamily.TrueTypeFile.GetFontFamilyName();
00607                         }
00608                         else
00609                         {
00610                             FontFamily = Font.FontFamily.FileName;
00611                         }
00612                     }
00613                     else
00614                     {
00615                         FontFamily = "resm:VectSharp.StandardFonts.?assembly=VectSharp#" +
      Font.FontFamily.TrueTypeFile.GetFontFamilyName();
00616                     }
00617                 }
00618             }
00619         }
00620 
00621         /*public (double Width, double Height) MeasureText(string text)
00622 {
00623 
00624 
00625 
00626 Avalonia.Media.FormattedText txt = new Avalonia.Media.FormattedText() { Text = text, Typeface = new
      Typeface(FontFamily), FontSize = Font.FontSize };
00627 
```

```
00628
00629
00630 return (txt.Bounds.Width, txt.Bounds.Height);
00631 }*/
00632
00633         private PathGeometry currentPath;
00634         private PathFigure currentFigure;
00635
00636         private bool figureInitialised = false;
00637
00638         public void MoveTo(double x, double y)
00639         {
00640             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00641
00642             if (figureInitialised)
00643             {
00644                 currentPath.Figures.Add(currentFigure);
00645             }
00646
00647             currentFigure = new PathFigure() { StartPoint = new Avalonia.Point(x, y), IsClosed = false
      };
00648             figureInitialised = true;
00649         }
00650
00651         public void LineTo(double x, double y)
00652         {
00653             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00654
00655             if (figureInitialised)
00656             {
00657                 currentFigure.Segments.Add(new Avalonia.Media.LineSegment() { Point = new
      Avalonia.Point(x, y) });
00658             }
00659             else
00660             {
00661                 currentFigure = new PathFigure() { StartPoint = new Avalonia.Point(x, y), IsClosed =
      false };
00662                 figureInitialised = true;
00663             }
00664         }
00665
00666         public void Rectangle(double x0, double y0, double width, double height)
00667         {
00668             Utils.CoerceNaNAndInfinityToZero(ref x0, ref y0, ref width, ref height);
00669
00670             if (currentFigure != null && figureInitialised)
00671             {
00672                 currentPath.Figures.Add(currentFigure);
00673             }
00674
00675             currentFigure = new PathFigure() { StartPoint = new Avalonia.Point(x0, y0), IsClosed =
      false };
00676             currentFigure.Segments.Add(new Avalonia.Media.LineSegment() { Point = new
      Avalonia.Point(x0 + width, y0) });
00677             currentFigure.Segments.Add(new Avalonia.Media.LineSegment() { Point = new
      Avalonia.Point(x0 + width, y0 + height) });
00678             currentFigure.Segments.Add(new Avalonia.Media.LineSegment() { Point = new
      Avalonia.Point(x0, y0 + height) });
00679             currentFigure.IsClosed = true;
00680
00681             currentPath.Figures.Add(currentFigure);
00682             figureInitialised = false;
00683         }
00684
00685         public void CubicBezierTo(double p1X, double p1Y, double p2X, double p2Y, double p3X, double
      p3Y)
00686         {
00687             Utils.CoerceNaNAndInfinityToZero(ref p1X, ref p1Y, ref p2X, ref p2Y, ref p3X, ref p3Y);
00688
00689             if (figureInitialised)
00690             {
00691                 currentFigure.Segments.Add(new Avalonia.Media.BezierSegment() { Point1 = new
      Avalonia.Point(p1X, p1Y), Point2 = new Avalonia.Point(p2X, p2Y), Point3 = new Avalonia.Point(p3X, p3Y)
      });
00692             }
00693             else
00694             {
00695                 currentFigure = new PathFigure() { StartPoint = new Avalonia.Point(p1X, p1Y), IsClosed
      = false };
00696                 figureInitialised = true;
00697             }
00698         }
00699
00700         public void Close()
00701         {
00702             currentFigure.IsClosed = true;
00703             currentPath.Figures.Add(currentFigure);
```

```
00704             figureInitialised = false;
00705         }
00706
00707         public void Stroke()
00708         {
00709             if (figureInitialised)
00710             {
00711                 currentFigure.IsClosed = false;
00712                 currentPath.Figures.Add(currentFigure);
00713             }
00714
00715             Avalonia.Media.Brush stroke = null;
00716
00717             if (this.StrokeStyle is SolidColourBrush solid)
00718             {
00719                 stroke = new SolidColorBrush(Color.FromArgb(StrokeAlpha, (byte)(solid.R * 255),
      (byte)(solid.G * 255), (byte)(solid.B * 255)));
00720             }
00721             else if (this.StrokeStyle is LinearGradientBrush linearGradient)
00722             {
00723                 stroke = linearGradient.ToLinearGradientBrush();
00724             }
00725             else if (this.StrokeStyle is RadialGradientBrush radialGradient)
00726             {
00727                 stroke = radialGradient.ToRadialGradientBrush(currentPath.Bounds.Width);
00728             }
00729
00730             Path pth = new Path() { Fill = null, Stroke = stroke, StrokeThickness = LineWidth,
      StrokeDashArray = new Avalonia.Collections.AvaloniaList<double>(LineDash.DashArray.Select(x => (x +
      (LineCap == LineCaps.Butt ?  0 :  LineWidth)) / LineWidth)), StrokeDashOffset = LineDash.Phase/
      LineWidth };
00731
00732             switch (LineCap)
00733             {
00734                 case LineCaps.Butt:
00735                     pth.StrokeLineCap = PenLineCap.Flat;
00736                     break;
00737                 case LineCaps.Round:
00738                     pth.StrokeLineCap = PenLineCap.Round;
00739                     break;
00740                 case LineCaps.Square:
00741                     pth.StrokeLineCap = PenLineCap.Square;
00742                     break;
00743             }
00744
00745             switch (LineJoin)
00746             {
00747                 case LineJoins.Bevel:
00748                     pth.StrokeJoin = PenLineJoin.Bevel;
00749                     break;
00750                 case LineJoins.Round:
00751                     pth.StrokeJoin = PenLineJoin.Round;
00752                     break;
00753                 case LineJoins.Miter:
00754                     pth.StrokeJoin = PenLineJoin.Miter;
00755                     break;
00756             }
00757
00758             pth.Data = currentPath;
00759
00760             pth.RenderTransform = new MatrixTransform(_transform.ToAvaloniaMatrix());
00761             pth.RenderTransformOrigin = new Avalonia.RelativePoint(0, 0,
      Avalonia.RelativeUnit.Absolute);
00762
00763             currControlElement.Children.Add(pth);
00764
00765             currentPath = new PathGeometry();
00766             currentFigure = new PathFigure() { IsClosed = false };
00767             figureInitialised = false;
00768
00769             if (!string.IsNullOrEmpty(Tag))
00770             {
00771                 if (TaggedActions.ContainsKey(Tag))
00772                 {
00773                     TaggedActions[Tag].DynamicInvoke(pth);
00774
00775                     if (removeTaggedActions)
00776                     {
00777                         TaggedActions.Remove(Tag);
00778                     }
00779                 }
00780             }
00781         }
00782
00783         public void Fill(FillRule fillRule)
00784         {
00785             if (figureInitialised)
```

```
00786             {
00787                 currentPath.Figures.Add(currentFigure);
00788             }
00789 
00790             Avalonia.Media.Brush fill = null;
00791 
00792             if (this.FillStyle is SolidColourBrush solid)
00793             {
00794                 fill = new SolidColorBrush(Color.FromArgb(FillAlpha, (byte)(solid.R * 255),
      (byte)(solid.G * 255), (byte)(solid.B * 255)));
00795             }
00796             else if (this.FillStyle is LinearGradientBrush linearGradient)
00797             {
00798                 fill = linearGradient.ToLinearGradientBrush();
00799             }
00800             else if (this.FillStyle is RadialGradientBrush radialGradient)
00801             {
00802                 fill = radialGradient.ToRadialGradientBrush(currentPath.Bounds.Width);
00803             }
00804 
00805             Path pth = new Path() { Fill = fill, Stroke = null };
00806 
00807             switch (fillRule)
00808             {
00809                 case FillRule.NonZeroWinding:
00810                     currentPath.FillRule = Avalonia.Media.FillRule.NonZero;
00811                     break;
00812 
00813                 case FillRule.EvenOdd:
00814                     currentPath.FillRule = Avalonia.Media.FillRule.EvenOdd;
00815                     break;
00816             }
00817 
00818             pth.Data = currentPath;
00819 
00820             pth.RenderTransform = new MatrixTransform(_transform.ToAvaloniaMatrix());
00821             pth.RenderTransformOrigin = new Avalonia.RelativePoint(0, 0,
      Avalonia.RelativeUnit.Absolute);
00822 
00823             currControlElement.Children.Add(pth);
00824 
00825             currentPath = new PathGeometry();
00826             currentFigure = new PathFigure() { IsClosed = false };
00827             figureInitialised = false;
00828 
00829             if (!string.IsNullOrEmpty(Tag))
00830             {
00831                 if (TaggedActions.ContainsKey(Tag))
00832                 {
00833                     TaggedActions[Tag].DynamicInvoke(pth);
00834 
00835                     if (removeTaggedActions)
00836                     {
00837                         TaggedActions.Remove(Tag);
00838                     }
00839                 }
00840             }
00841         }
00842 
00843         public void SetClippingPath()
00844         {
00845             if (figureInitialised)
00846             {
00847                 currentPath.Figures.Add(currentFigure);
00848             }
00849 
00850             Avalonia.Controls.Canvas newControlElement = new Avalonia.Controls.Canvas();
00851 
00852             newControlElement.Clip = currentPath;
00853 
00854             newControlElement.RenderTransformOrigin = new Avalonia.RelativePoint(0, 0,
      Avalonia.RelativeUnit.Absolute);
00855             newControlElement.RenderTransform = new MatrixTransform(_transform.ToAvaloniaMatrix());
00856 
00857             _transform = new double[3, 3];
00858 
00859             _transform[0, 0] = 1;
00860             _transform[1, 1] = 1;
00861             _transform[2, 2] = 1;
00862 
00863             currControlElement.Children.Add(newControlElement);
00864             currControlElement = newControlElement;
00865 
00866             currentPath = new PathGeometry();
00867             currentFigure = new PathFigure() { IsClosed = false };
00868             figureInitialised = false;
00869         }
```

```
00870
00871
00872         public void DrawRasterImage(int sourceX, int sourceY, int sourceWidth, int sourceHeight,
      double destinationX, double destinationY, double destinationWidth, double destinationHeight,
      RasterImage image)
00873         {
00874             Utils.CoerceNaNAndInfinityToZero(ref destinationX, ref destinationY, ref destinationWidth,
      ref destinationHeight);
00875
00876             Image img = new Image() { Source = new CroppedBitmap(new Bitmap(image.PNGStream), new
      Avalonia.PixelRect(sourceX, sourceY, sourceWidth, sourceHeight)), Width = destinationWidth, Height =
      destinationHeight };
00877
00878             if (image.Interpolate)
00879             {
00880                 RenderOptions.SetBitmapInterpolationMode(img,
      Avalonia.Media.Imaging.BitmapInterpolationMode.HighQuality);
00881             }
00882             else
00883             {
00884                 RenderOptions.SetBitmapInterpolationMode(img,
      Avalonia.Media.Imaging.BitmapInterpolationMode.None);
00885             }
00886
00887             double[,] transf = MatrixUtils.Translate(_transform, destinationX, destinationY);
00888             img.RenderTransform = new MatrixTransform(transf.ToAvaloniaMatrix());
00889             img.RenderTransformOrigin = new Avalonia.RelativePoint(0, 0,
      Avalonia.RelativeUnit.Absolute);
00890
00891             currControlElement.Children.Add(img);
00892
00893             if (!string.IsNullOrEmpty(Tag))
00894             {
00895                 if (TaggedActions.ContainsKey(Tag))
00896                 {
00897                     TaggedActions[Tag].DynamicInvoke(img);
00898
00899                     if (removeTaggedActions)
00900                     {
00901                         TaggedActions.Remove(Tag);
00902                     }
00903                 }
00904             }
00905         }
00906
00907         public void DrawFilteredGraphics(Graphics graphics, IFilter filter)
00908         {
00909             if (this._filterOption.Operation == FilterOption.FilterOperations.RasteriseAllWithSkia)
00910             {
00911                 double scale = this._filterOption.RasterisationResolution;
00912
00913                 Rectangle bounds = graphics.GetBounds();
00914
00915                 bounds = new Rectangle(bounds.Location.X - filter.TopLeftMargin.X, bounds.Location.Y -
      filter.TopLeftMargin.Y, bounds.Size.Width + filter.TopLeftMargin.X + filter.BottomRightMargin.X,
      bounds.Size.Height + filter.TopLeftMargin.Y + filter.BottomRightMargin.Y);
00916
00917                 if (bounds.Size.Width > 0 && bounds.Size.Height > 0)
00918                 {
00919                     if (!this._filterOption.RasterisationResolutionRelative)
00920                     {
00921                         scale = scale / Math.Min(bounds.Size.Width, bounds.Size.Height);
00922                     }
00923
00924                     RasterImage rasterised = SKRenderContextInterpreter.Rasterise(graphics, bounds,
      scale, true);
00925                     RasterImage filtered = null;
00926
00927                     if (filter is IFilterWithRasterisableParameter filterWithRastParam)
00928                     {
00929                         filterWithRastParam.RasteriseParameter(SKRenderContextInterpreter.Rasterise,
      scale);
00930                     }
00931
00932                     if (filter is ILocationInvariantFilter locInvFilter)
00933                     {
00934                         filtered = locInvFilter.Filter(rasterised, scale);
00935                     }
00936                     else if (filter is IFilterWithLocation filterWithLoc)
00937                     {
00938                         filtered = filterWithLoc.Filter(rasterised, bounds, scale);
00939                     }
00940
00941                     if (filtered != null)
00942                     {
00943                         rasterised.Dispose();
00944
```

```
00945                         DrawRasterImage(0, 0, filtered.Width, filtered.Height, bounds.Location.X,
      bounds.Location.Y, bounds.Size.Width, bounds.Size.Height, filtered);
00946                     }
00947                 }
00948             }
00949             else if (this._filterOption.Operation ==
      FilterOption.FilterOperations.RasteriseAllWithVectSharp)
00950             {
00951                 double scale = this._filterOption.RasterisationResolution;
00952 
00953                 Rectangle bounds = graphics.GetBounds();
00954 
00955                 bounds = new Rectangle(bounds.Location.X - filter.TopLeftMargin.X, bounds.Location.Y -
      filter.TopLeftMargin.Y, bounds.Size.Width + filter.TopLeftMargin.X + filter.BottomRightMargin.X,
      bounds.Size.Height + filter.TopLeftMargin.Y + filter.BottomRightMargin.Y);
00956 
00957                 if (bounds.Size.Width > 0 && bounds.Size.Height > 0)
00958                 {
00959                     if (!this._filterOption.RasterisationResolutionRelative)
00960                     {
00961                         scale = scale / Math.Min(bounds.Size.Width, bounds.Size.Height);
00962                     }
00963 
00964                     if (graphics.TryRasterise(bounds, scale, true, out RasterImage rasterised))
00965                     {
00966                         RasterImage filtered = null;
00967 
00968                         if (filter is IFilterWithRasterisableParameter filterWithRastParam)
00969                         {
00970 
      filterWithRastParam.RasteriseParameter(SKRenderContextInterpreter.Rasterise, scale);
00971                         }
00972 
00973                         if (filter is ILocationInvariantFilter locInvFilter)
00974                         {
00975                             filtered = locInvFilter.Filter(rasterised, scale);
00976                         }
00977                         else if (filter is IFilterWithLocation filterWithLoc)
00978                         {
00979                             filtered = filterWithLoc.Filter(rasterised, bounds, scale);
00980                         }
00981 
00982                         if (filtered != null)
00983                         {
00984                             rasterised.Dispose();
00985 
00986                             DrawRasterImage(0, 0, filtered.Width, filtered.Height, bounds.Location.X,
      bounds.Location.Y, bounds.Size.Width, bounds.Size.Height, filtered);
00987                         }
00988                     }
00989                     else
00990                     {
00991                         throw new NotImplementedException(@"The filter could not be rasterised!  You
      can avoid this error by doing one of the following:
00992 • Add a reference to VectSharp.Raster or VectSharp.Raster.ImageSharp (you may also need to add a using
      directive somewhere to force the assembly to be loaded).
00993 • Provide your own implementation of Graphics.RasterisationMethod.
00994 • Set the FilterOption.Operation to ""RasteriseAllWithSkia"", ""IgnoreAll"" or ""SkipAll"".");
00995                     }
00996                 }
00997             }
00998             else if (this._filterOption.Operation == FilterOption.FilterOperations.IgnoreAll)
00999             {
01000                 graphics.CopyToIGraphicsContext(this);
01001             }
01002             else
01003             {
01004 
01005             }
01006         }
01007     }
01008 
01009     internal class RenderCanvas :  Control
01010     {
01011         private List<RenderAction> RenderActions;
01012         private List<RenderAction> TaggedRenderActions;
01013 
01014         private SolidColorBrush BackgroundBrush;
01015 
01016         static Avalonia.Point Origin = new Avalonia.Point(0, 0);
01017 
01018         public Dictionary<string, (IImage, bool)> Images;
01019 
01020         public void BringToFront(RenderAction action)
01021         {
01022             this.RenderActions.Remove(action);
01023             this.RenderActions.Add(action);
```

```
01024
01025             if (!string.IsNullOrEmpty(action.Tag))
01026             {
01027                 int index = this.TaggedRenderActions.IndexOf(action);
01028                 if (index >= 0)
01029                 {
01030                     this.TaggedRenderActions.RemoveAt(index);
01031                     this.TaggedRenderActions.Insert(0, action);
01032                 }
01033             }
01034         }
01035
01036         public void SendToBack(RenderAction action)
01037         {
01038             this.RenderActions.Remove(action);
01039             this.RenderActions.Insert(0, action);
01040
01041             if (!string.IsNullOrEmpty(action.Tag))
01042             {
01043                 int index = this.TaggedRenderActions.IndexOf(action);
01044                 if (index >= 0)
01045                 {
01046                     this.TaggedRenderActions.RemoveAt(index);
01047                     this.TaggedRenderActions.Add(action);
01048                 }
01049             }
01050         }
01051
01052         public RenderCanvas(Graphics content, Colour backgroundColour, double width, double height,
      Dictionary<string, Delegate> taggedActions, bool removeTaggedActionsAfterExecution,
      AvaloniaContextInterpreter.TextOptions textOption, FilterOption filterOption)
01053         {
01054             this.BackgroundBrush = new SolidColorBrush(Color.FromArgb((byte)(backgroundColour.A *
      255), (byte)(backgroundColour.R * 255), (byte)(backgroundColour.G * 255), (byte)(backgroundColour.B *
      255)));
01055             this.Width = width;
01056             this.Height = height;
01057             this.Images = new Dictionary<string, (IImage, bool)>();
01058             AvaloniaDrawingContext ctx = new AvaloniaDrawingContext(this.Width, this.Height,
      removeTaggedActionsAfterExecution, textOption, this.Images, filterOption);
01059             foreach (KeyValuePair<string, Delegate> action in taggedActions)
01060             {
01061                 ctx.TaggedActions.Add(action.Key, (Func<RenderAction,
      IEnumerable<RenderAction>)action.Value);
01062             }
01063
01064             content.CopyToIGraphicsContext(ctx);
01065             this.RenderActions = ctx.RenderActions;
01066
01067             this.TaggedRenderActions = new List<RenderAction>();
01068
01069             for (int i = this.RenderActions.Count - 1; i >= 0; i--)
01070             {
01071                 RenderActions[i].InternalParent = this;
01072                 if (!string.IsNullOrEmpty(this.RenderActions[i].Tag))
01073                 {
01074                     TaggedRenderActions.Add(this.RenderActions[i]);
01075                 }
01076             }
01077
01078             this.PointerPressed += PointerPressedAction;
01079             this.PointerReleased += PointerReleasedAction;
01080             this.PointerMoved += PointerMoveAction;
01081             this.PointerExited += PointerLeaveAction;
01082         }
01083
01084
01085         private int CurrentPressedAction = -1;
01086         private void PointerPressedAction(object sender, Avalonia.Input.PointerPressedEventArgs e)
01087         {
01088             Avalonia.Point position = e.GetPosition(this);
01089
01090             for (int i = 0; i < TaggedRenderActions.Count; i++)
01091             {
01092                 if (TaggedRenderActions[i].ClippingPath == null ||
      TaggedRenderActions[i].ClippingPath.FillContains(position))
01093                 {
01094                     Avalonia.Point localPosition =
      position.Transform(TaggedRenderActions[i].InverseTransform);
01095
01096                     if (TaggedRenderActions[i].ActionType == RenderAction.ActionTypes.Path)
01097                     {
01098                         if ((TaggedRenderActions[i].Fill != null &&
      TaggedRenderActions[i].Geometry.FillContains(localPosition)) ||
      TaggedRenderActions[i].Geometry.StrokeContains(TaggedRenderActions[i].Stroke, localPosition))
01099                         {
01100                             TaggedRenderActions[i].FirePointerPressed(e);
```

```
01101                           CurrentPressedAction = i;
01102                           break;
01103                       }
01104                   }
01105                   else if (TaggedRenderActions[i].ActionType == RenderAction.ActionTypes.Text)
01106                   {
01107                       if (TaggedRenderActions[i].Fill != null &&
      TaggedRenderActions[i].Layout.HitTestPoint(localPosition).IsInside)
01108                       {
01109                           TaggedRenderActions[i].FirePointerPressed(e);
01110                           CurrentPressedAction = i;
01111                           break;
01112                       }
01113                   }
01114                   else if (TaggedRenderActions[i].ActionType ==
      RenderAction.ActionTypes.RasterImage)
01115                   {
01116                       if (TaggedRenderActions[i].ImageDestination.Value.Contains(localPosition))
01117                       {
01118                           TaggedRenderActions[i].FirePointerPressed(e);
01119                           CurrentPressedAction = i;
01120                           break;
01121                       }
01122                   }
01123               }
01124           }
01125       }
01126
01127       private void PointerReleasedAction(object sender, Avalonia.Input.PointerReleasedEventArgs e)
01128       {
01129           if (CurrentPressedAction >= 0)
01130           {
01131               TaggedRenderActions[CurrentPressedAction].FirePointerReleased(e);
01132               CurrentPressedAction = -1;
01133           }
01134           else
01135           {
01136               Avalonia.Point position = e.GetPosition(this);
01137
01138               for (int i = 0; i < TaggedRenderActions.Count; i++)
01139               {
01140                   if (TaggedRenderActions[i].ClippingPath == null ||
      TaggedRenderActions[i].ClippingPath.FillContains(position))
01141                   {
01142                       Avalonia.Point localPosition =
      position.Transform(TaggedRenderActions[i].InverseTransform);
01143
01144                       if (TaggedRenderActions[i].ActionType == RenderAction.ActionTypes.Path)
01145                       {
01146                           if ((TaggedRenderActions[i].Fill != null &&
      TaggedRenderActions[i].Geometry.FillContains(localPosition)) ||
      TaggedRenderActions[i].Geometry.StrokeContains(TaggedRenderActions[i].Stroke, localPosition))
01147                           {
01148                               TaggedRenderActions[i].FirePointerReleased(e);
01149                               break;
01150                           }
01151                       }
01152                       else if (TaggedRenderActions[i].ActionType == RenderAction.ActionTypes.Text)
01153                       {
01154                           if (TaggedRenderActions[i].Fill != null &&
      TaggedRenderActions[i].Layout.HitTestPoint(localPosition).IsInside)
01155                           {
01156                               TaggedRenderActions[i].FirePointerReleased(e);
01157                               break;
01158                           }
01159                       }
01160                       else if (TaggedRenderActions[i].ActionType ==
      RenderAction.ActionTypes.RasterImage)
01161                       {
01162                           if (TaggedRenderActions[i].ImageDestination.Value.Contains(localPosition))
01163                           {
01164                               TaggedRenderActions[i].FirePointerReleased(e);
01165                               break;
01166                           }
01167                       }
01168                   }
01169               }
01170           }
01171       }
01172
01173       private int CurrentOverAction = -1;
01174       private void PointerMoveAction(object sender, Avalonia.Input.PointerEventArgs e)
01175       {
01176           Avalonia.Point position = e.GetPosition(this);
01177
01178           bool found = false;
01179
```

```
01180             for (int i = 0; i < TaggedRenderActions.Count; i++)
01181             {
01182                 if (TaggedRenderActions[i].ClippingPath == null ||
      TaggedRenderActions[i].ClippingPath.FillContains(position))
01183                 {
01184                     Avalonia.Point localPosition =
      position.Transform(TaggedRenderActions[i].InverseTransform);
01185 
01186                     if (TaggedRenderActions[i].ActionType == RenderAction.ActionTypes.Path)
01187                     {
01188                         if ((TaggedRenderActions[i].Fill != null &&
      TaggedRenderActions[i].Geometry.FillContains(localPosition)) ||
      TaggedRenderActions[i].Geometry.StrokeContains(TaggedRenderActions[i].Stroke, localPosition))
01189                         {
01190                             found = true;
01191 
01192                             if (CurrentOverAction != i)
01193                             {
01194                                 if (CurrentOverAction >= 0)
01195                                 {
01196                                     TaggedRenderActions[CurrentOverAction].FirePointerExited(e);
01197                                 }
01198                                 CurrentOverAction = i;
01199                                 TaggedRenderActions[CurrentOverAction].FirePointerEntered(e);
01200                             }
01201 
01202                             break;
01203                         }
01204                     }
01205                     else if (TaggedRenderActions[i].ActionType == RenderAction.ActionTypes.Text)
01206                     {
01207                         if (TaggedRenderActions[i].Fill != null &&
      TaggedRenderActions[i].Layout.HitTestPoint(localPosition).IsInside)
01208                         {
01209                             found = true;
01210 
01211                             if (CurrentOverAction != i)
01212                             {
01213                                 if (CurrentOverAction >= 0)
01214                                 {
01215                                     TaggedRenderActions[CurrentOverAction].FirePointerExited(e);
01216                                 }
01217                                 CurrentOverAction = i;
01218                                 TaggedRenderActions[CurrentOverAction].FirePointerEntered(e);
01219                             }
01220 
01221                             break;
01222                         }
01223                     }
01224                     else if (TaggedRenderActions[i].ActionType ==
      RenderAction.ActionTypes.RasterImage)
01225                     {
01226                         if (TaggedRenderActions[i].ImageDestination.Value.Contains(localPosition))
01227                         {
01228                             found = true;
01229 
01230                             if (CurrentOverAction != i)
01231                             {
01232                                 if (CurrentOverAction >= 0)
01233                                 {
01234                                     TaggedRenderActions[CurrentOverAction].FirePointerExited(e);
01235                                 }
01236                                 CurrentOverAction = i;
01237                                 TaggedRenderActions[CurrentOverAction].FirePointerEntered(e);
01238                             }
01239 
01240                             break;
01241                         }
01242                     }
01243                 }
01244             }
01245 
01246             if (!found)
01247             {
01248                 if (CurrentOverAction >= 0)
01249                 {
01250                     TaggedRenderActions[CurrentOverAction].FirePointerExited(e);
01251                 }
01252                 CurrentOverAction = -1;
01253             }
01254         }
01255 
01256         private void PointerLeaveAction(object sender, Avalonia.Input.PointerEventArgs e)
01257         {
01258             if (CurrentOverAction >= 0)
01259             {
01260                 TaggedRenderActions[CurrentOverAction].FirePointerExited(e);
```

```
01261             }
01262             CurrentOverAction = -1;
01263         }
01264 
01265 
01266         public override void Render(DrawingContext context)
01267         {
01268             context.FillRectangle(this.BackgroundBrush, new Avalonia.Rect(0, 0, Width, Height));
01269 
01270             foreach (RenderAction act in this.RenderActions)
01271             {
01272                 if (act.ActionType == RenderAction.ActionTypes.Path)
01273                 {
01274                     DrawingContext.PushedState?  state = null;
01275 
01276                     if (act.ClippingPath != null)
01277                     {
01278                         //Random draw operation needed due to
      https://github.com/AvaloniaUI/Avalonia/issues/4408
01279                         context.DrawGeometry(null, new Pen(Brushes.Transparent), act.ClippingPath);
01280                         state = context.PushGeometryClip(act.ClippingPath);
01281                     }
01282 
01283                     using (context.PushTransform(act.Transform))
01284                     {
01285                         context.DrawGeometry(act.Fill, act.Stroke, act.Geometry);
01286                     }
01287 
01288                     if (state != null)
01289                     {
01290                         state?.Dispose();
01291                         //Random draw operation needed due to
      https://github.com/AvaloniaUI/Avalonia/issues/4408
01292                         context.DrawGeometry(null, new Pen(Brushes.Transparent), act.ClippingPath);
01293                     }
01294                 }
01295                 else if (act.ActionType == RenderAction.ActionTypes.Text)
01296                 {
01297                     DrawingContext.PushedState?  state = null;
01298 
01299                     if (act.ClippingPath != null)
01300                     {
01301                         //Random draw operation needed due to
      https://github.com/AvaloniaUI/Avalonia/issues/4408
01302                         context.DrawGeometry(null, new Pen(Brushes.Transparent), act.ClippingPath);
01303                         state = context.PushGeometryClip(act.ClippingPath);
01304                     }
01305 
01306                     using (context.PushTransform(act.Transform))
01307                     {
01308                         context.DrawText(act.Text, Origin);
01309                     }
01310 
01311                     if (state != null)
01312                     {
01313                         state?.Dispose();
01314                         //Random draw operation needed due to
      https://github.com/AvaloniaUI/Avalonia/issues/4408
01315                         context.DrawGeometry(null, new Pen(Brushes.Transparent), act.ClippingPath);
01316                     }
01317                 }
01318                 else if (act.ActionType == RenderAction.ActionTypes.RasterImage)
01319                 {
01320                     DrawingContext.PushedState?  state = null;
01321 
01322                     if (act.ClippingPath != null)
01323                     {
01324                         //Random draw operation needed due to
      https://github.com/AvaloniaUI/Avalonia/issues/4408
01325                         context.DrawGeometry(null, new Pen(Brushes.Transparent), act.ClippingPath);
01326                         state = context.PushGeometryClip(act.ClippingPath);
01327                     }
01328 
01329                     (IImage, bool) image = Images[act.ImageId];
01330 
01331                     using (context.PushTransform(act.Transform))
01332                     {
01333                         using (context.PushRenderOptions(new RenderOptions() { BitmapInterpolationMode
      = image.Item2 ?  Avalonia.Media.Imaging.BitmapInterpolationMode.HighQuality :
      Avalonia.Media.Imaging.BitmapInterpolationMode.None }))
01334                         {
01335                             context.DrawImage(image.Item1, act.ImageSource.Value,
      act.ImageDestination.Value);
01336                         }
01337                     }
01338 
01339                     if (state != null)
```

```
01340                     {
01341                         state?.Dispose();
01342                         //Random draw operation needed due to
      https://github.com/AvaloniaUI/Avalonia/issues/4408
01343                         context.DrawGeometry(null, new Pen(Brushes.Transparent), act.ClippingPath);
01344                     }
01345                 }
01346             }
01347         }
01348     }
01349
01350
01351 /// <summary>
01352 /// Represents a light-weight rendering action.
01353 /// </summary>
01354     public class RenderAction
01355     {
01356 /// <summary>
01357 /// Types of rendering actions.
01358 /// </summary>
01359         public enum ActionTypes
01360         {
01361 /// <summary>
01362 /// The render action represents a path object.
01363 /// </summary>
01364             Path,
01365
01366 /// <summary>
01367 /// The render action represents a text object.
01368 /// </summary>
01369             Text,
01370
01371 /// <summary>
01372 /// The render action represents a raster image.
01373 /// </summary>
01374             RasterImage
01375         }
01376
01377 /// <summary>
01378 /// Type of the rendering action.
01379 /// </summary>
01380         public ActionTypes ActionType { get; private set; }
01381
01382 /// <summary>
01383 /// Geometry that needs to be rendered (null if the action type is <see cref="ActionTypes.Text"/>).
      If you change this, you need to invalidate the <see cref="Parent"/>'s visual.
01384 /// </summary>
01385         public Geometry Geometry { get; set; }
01386
01387 /// <summary>
01388 /// Text that needs to be rendered (null if the action type is <see cref="ActionTypes.Path"/>).  If
      you change this, you need to invalidate the <see cref="Parent"/>'s visual and
01389 /// you should also change the <see cref="Layout"/> property.
01390 /// </summary>
01391         public Avalonia.Media.FormattedText Text { get; set; }
01392
01393 /// <summary>
01394 /// Text layout, used for hit testing (null if the action type is <see cref="ActionTypes.Path"/>).  If
      you change this, you need to invalidate the <see cref="Parent"/>'s visual and
01395 /// you should also change the <see cref="Text"/> property.
01396 /// </summary>
01397         public Avalonia.Media.TextFormatting.TextLayout Layout { get; set; }
01398
01399 /// <summary>
01400 /// Rendering stroke (null if the action type is <see cref="ActionTypes.Text"/> or if the rendered
      action only has a <see cref="Fill"/>).  If you change this, you need to invalidate the <see
      cref="Parent"/>'s visual.
01401 /// </summary>
01402         public Pen Stroke { get; set; }
01403
01404
01405         private IBrush fill;
01406
01407 /// <summary>
01408 /// Rendering fill (null if the rendered action only has a <see cref="Stroke"/>).  If you change this,
      you need to invalidate the <see cref="Parent"/>'s visual.
01409 /// </summary>
01410         public IBrush Fill
01411         {
01412             get
01413             {
01414                 return fill;
01415             }
01416             set
01417             {
01418                 fill = value;
01419
```

```
01420                 if (Text != null)
01421                 {
01422                     Text.SetForegroundBrush(fill);
01423                 }
01424             }
01425         }
01426 
01427 /// <summary>
01428 /// Univocal identifier of the image that needs to be drawn.
01429 /// </summary>
01430         public string ImageId { get; set; }
01431 
01432 /// <summary>
01433 /// The source rectangle of the image.
01434 /// </summary>
01435         public Avalonia.Rect?  ImageSource { get; set; }
01436 
01437 /// <summary>
01438 /// The destination rectangle of the image.
01439 /// </summary>
01440         public Avalonia.Rect?  ImageDestination { get; set; }
01441 
01442 /// <summary>
01443 /// The current clipping path.
01444 /// </summary>
01445         public Geometry ClippingPath { get; set; }
01446 
01447         private Avalonia.Matrix _transform = Avalonia.Matrix.Identity;
01448 
01449 /// <summary>
01450 /// Inverse transformation matrix.
01451 /// </summary>
01452         public Avalonia.Matrix InverseTransform { get; private set; } = Avalonia.Matrix.Identity;
01453 
01454 /// <summary>
01455 /// Rendering transformation matrix.  If you change this, you need to invalidate the <see
      cref="Parent"/>'s visual.
01456 /// </summary>
01457         public Avalonia.Matrix Transform
01458         {
01459             get { return _transform; }
01460             set
01461             {
01462                 _transform = value;
01463                 InverseTransform = _transform.Invert();
01464             }
01465         }
01466 
01467 /// <summary>
01468 /// A tag to access the <see cref="RenderAction"/>.
01469 /// </summary>
01470         public string Tag { get; set; }
01471 
01472         internal RenderCanvas InternalParent { get; set; }
01473 
01474 /// <summary>
01475 /// The container of this <see cref="RenderAction"/>.
01476 /// </summary>
01477         public Control Parent
01478         {
01479             get
01480             {
01481                 return InternalParent;
01482             }
01483         }
01484 
01485 /// <summary>
01486 /// Raised when the pointer enters the area covered by the <see cref="RenderAction"/>.
01487 /// </summary>
01488         public event EventHandler<Avalonia.Input.PointerEventArgs> PointerEntered;
01489 
01490 /// <summary>
01491 /// Raised when the pointer leaves the area covered by the <see cref="RenderAction"/>.
01492 /// </summary>
01493         public event EventHandler<Avalonia.Input.PointerEventArgs> PointerExited;
01494 
01495 /// <summary>
01496 /// Raised when the pointer is pressed while over the area covered by the <see cref="RenderAction"/>.
01497 /// </summary>
01498         public event EventHandler<Avalonia.Input.PointerPressedEventArgs> PointerPressed;
01499 
01500 /// <summary>
01501 /// Raised when the pointer is released after a <see cref="PointerPressed"/> event.
01502 /// </summary>
01503         public event EventHandler<Avalonia.Input.PointerReleasedEventArgs> PointerReleased;
01504 
01505 
```

```
01506         internal void FirePointerEntered(Avalonia.Input.PointerEventArgs e)
01507         {
01508             this.PointerEntered?.Invoke(this, e);
01509         }
01510 
01511         internal void FirePointerExited(Avalonia.Input.PointerEventArgs e)
01512         {
01513             this.PointerExited?.Invoke(this, e);
01514         }
01515 
01516         internal void FirePointerPressed(Avalonia.Input.PointerPressedEventArgs e)
01517         {
01518             this.PointerPressed?.Invoke(this, e);
01519         }
01520 
01521         internal void FirePointerReleased(Avalonia.Input.PointerReleasedEventArgs e)
01522         {
01523             this.PointerReleased?.Invoke(this, e);
01524         }
01525 
01526         private RenderAction()
01527         {
01528 
01529         }
01530 
01531 /// <summary>
01532 /// Creates a new <see cref="RenderAction"/> representing a path.
01533 /// </summary>
01534 /// <param name="geometry">The geometry to be rendered.</param>
01535 /// <param name="stroke">The stroke of the path (can be null).</param>
01536 /// <param name="fill">The fill of the path (can be null).</param>
01537 /// <param name="transform">The transform that will be applied to the path.</param>
01538 /// <param name="clippingPath">The clipping path.</param>
01539 /// <param name="tag">A tag to access the <see cref="RenderAction"/>.  If this is null this <see
      cref="RenderAction"/> is not visible in the hit test.</param>
01540 /// <returns>A new <see cref="RenderAction"/> representing a path.</returns>
01541         public static RenderAction PathAction(Geometry geometry, Pen stroke, IBrush fill,
      Avalonia.Matrix transform, Geometry clippingPath, string tag = null)
01542         {
01543             return new RenderAction()
01544             {
01545                 ActionType = ActionTypes.Path,
01546                 Geometry = geometry,
01547                 Stroke = stroke,
01548                 Fill = fill,
01549                 Transform = transform,
01550                 ClippingPath = clippingPath,
01551                 Tag = tag
01552             };
01553         }
01554 
01555 /// <summary>
01556 /// Creates a new <see cref="RenderAction"/> representing text.
01557 /// </summary>
01558 /// <param name="text">The text to be rendered.</param>
01559 /// <param name="layout">The text layout used for hit testing.</param>
01560 /// <param name="fill">The fill of the text (can be null).</param>
01561 /// <param name="transform">The transform that will be applied to the text.</param>
01562 /// <param name="clippingPath">The clipping path.</param>
01563 /// <param name="tag">A tag to access the <see cref="RenderAction"/>.  If this is null this <see
      cref="RenderAction"/> is not visible in the hit test.</param>
01564 /// <returns>A new <see cref="RenderAction"/> representing text.</returns>
01565         public static RenderAction TextAction(Avalonia.Media.FormattedText text,
      Avalonia.Media.TextFormatting.TextLayout layout, IBrush fill, Avalonia.Matrix transform, Geometry
      clippingPath, string tag = null)
01566         {
01567             return new RenderAction()
01568             {
01569                 ActionType = ActionTypes.Text,
01570                 Text = text,
01571                 Stroke = null,
01572                 Fill = fill,
01573                 Transform = transform,
01574                 ClippingPath = clippingPath,
01575                 Tag = tag,
01576                 Layout = layout
01577             };
01578         }
01579 
01580 /// <summary>
01581 /// Creates a new <see cref="RenderAction"/> representing an image.
01582 /// </summary>
01583 /// <param name="imageId">The univocal identifier of the image to draw.</param>
01584 /// <param name="sourceRect">The source rectangle of the image.</param>
01585 /// <param name="destinationRect">The destination rectangle of the image.</param>
01586 /// <param name="transform">The transform that will be applied to the image.</param>
01587 /// <param name="clippingPath">The clipping path.</param>
```

```
01588 /// <param name="tag">A tag to access the <see cref="RenderAction"/>.  If this is null this <see
      cref="RenderAction"/> is not visible in the hit test.</param>
01589 /// <returns>A new <see cref="RenderAction"/> representing an image.</returns>
01590         public static RenderAction ImageAction(string imageId, Avalonia.Rect sourceRect, Avalonia.Rect
      destinationRect, Avalonia.Matrix transform, Geometry clippingPath, string tag = null)
01591         {
01592             return new RenderAction()
01593             {
01594                 ActionType = ActionTypes.RasterImage,
01595                 ImageId = imageId,
01596                 ImageSource = sourceRect,
01597                 ImageDestination = destinationRect,
01598                 Transform = transform,
01599                 ClippingPath = clippingPath,
01600                 Tag = tag
01601             };
01602         }
01603
01604 /// <summary>
01605 /// Brings the render action to the front of the rendering queue.  This method can only be invoked
      after the output has been fully initialised.
01606 /// </summary>
01607         public void BringToFront()
01608         {
01609             this.InternalParent.BringToFront(this);
01610         }
01611
01612 /// <summary>
01613 /// Brings the render action to the back of the rendering queue.  This method can only be invoked
      after the output has been fully initialised.
01614 /// </summary>
01615         public void SendToBack()
01616         {
01617             this.InternalParent.SendToBack(this);
01618         }
01619     }
01620
01621
01622     internal class AvaloniaDrawingContext :  IGraphicsContext
01623     {
01624         public Dictionary<string, Func<RenderAction, IEnumerable<RenderAction>> TaggedActions { get;
      set; } = new Dictionary<string, Func<RenderAction, IEnumerable<RenderAction>>();
01625
01626         private bool removeTaggedActions = true;
01627
01628         public string Tag { get; set; }
01629
01630         AvaloniaContextInterpreter.TextOptions _textOption;
01631
01632         private Dictionary<string, (IImage, bool)> Images;
01633
01634         private Geometry _clippingPath;
01635         private Stack<Geometry> clippingPaths;
01636         private FilterOption _filterOption;
01637
01638         public AvaloniaDrawingContext(double width, double height, bool
      removeTaggedActionsAfterExecution, AvaloniaContextInterpreter.TextOptions textOption,
      Dictionary<string, (IImage, bool)> images, FilterOption filterOption)
01639         {
01640             this.Images = images;
01641
01642             currentPath = new PathGeometry();
01643             currentFigure = new PathFigure() { IsClosed = false };
01644             figureInitialised = false;
01645
01646             RenderActions = new List<RenderAction>();
01647             removeTaggedActions = removeTaggedActionsAfterExecution;
01648
01649             Width = width;
01650             Height = height;
01651
01652             _transform = new double[3, 3];
01653
01654             _transform[0, 0] = 1;
01655             _transform[1, 1] = 1;
01656             _transform[2, 2] = 1;
01657
01658             states = new Stack<double[,]>();
01659
01660             _textOption = textOption;
01661
01662             _clippingPath = null;
01663             clippingPaths = new Stack<Geometry>();
01664             clippingPaths.Push(_clippingPath);
01665
01666             _filterOption = filterOption;
01667         }
```

```
01668
01669         public List<RenderAction> RenderActions { get; set; }
01670
01671         public double Width { get; private set; }
01672         public double Height { get; private set; }
01673
01674         public void Translate(double x, double y)
01675         {
01676             _transform = MatrixUtils.Translate(_transform, x, y);
01677
01678             currentPath = new PathGeometry();
01679             currentFigure = new PathFigure() { IsClosed = false };
01680             figureInitialised = false;
01681         }
01682
01683         public TextBaselines TextBaseline { get; set; }
01684
01685         private void PathText(string text, double x, double y)
01686         {
01687             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
01688
01689             GraphicsPath textPath = new GraphicsPath().AddText(x, y, text, Font, TextBaseline);
01690
01691             for (int j = 0; j < textPath.Segments.Count; j++)
01692             {
01693                 switch (textPath.Segments[j].Type)
01694                 {
01695                     case VectSharp.SegmentType.Move:
01696                         this.MoveTo(textPath.Segments[j].Point.X, textPath.Segments[j].Point.Y);
01697                         break;
01698                     case VectSharp.SegmentType.Line:
01699                         this.LineTo(textPath.Segments[j].Point.X, textPath.Segments[j].Point.Y);
01700                         break;
01701                     case VectSharp.SegmentType.CubicBezier:
01702                         this.CubicBezierTo(textPath.Segments[j].Points[0].X,
      textPath.Segments[j].Points[0].Y, textPath.Segments[j].Points[1].X, textPath.Segments[j].Points[1].Y,
      textPath.Segments[j].Points[2].X, textPath.Segments[j].Points[2].Y);
01703                         break;
01704                     case VectSharp.SegmentType.Close:
01705                         this.Close();
01706                         break;
01707                 }
01708             }
01709         }
01710
01711         public void StrokeText(string text, double x, double y)
01712         {
01713             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
01714
01715             PathText(text, x, y);
01716             Stroke();
01717         }
01718
01719         public void FillText(string text, double x, double y)
01720         {
01721             if (!Font.EnableKerning)
01722             {
01723                 FillSimpleText(text, x, y);
01724             }
01725             else
01726             {
01727                 List<(string, Point)> tSpans = new List<(string, Point)>();
01728
01729                 System.Text.StringBuilder currentRun = new System.Text.StringBuilder();
01730                 Point currentKerning = new Point();
01731
01732                 Point currentGlyphPlacementDelta = new Point();
01733                 Point currentGlyphAdvanceDelta = new Point();
01734                 Point nextGlyphPlacementDelta = new Point();
01735                 Point nextGlyphAdvanceDelta = new Point();
01736
01737                 for (int i = 0; i < text.Length; i++)
01738                 {
01739                     if (i < text.Length - 1)
01740                     {
01741                         currentGlyphPlacementDelta = nextGlyphPlacementDelta;
01742                         currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
01743                         nextGlyphAdvanceDelta = new Point();
01744                         nextGlyphPlacementDelta = new Point();
01745
01746                         TrueTypeFile.PairKerning kerning =
      Font.FontFamily.TrueTypeFile.Get1000EmKerning(text[i], text[i + 1]);
01747
01748                         if (kerning != null)
01749                         {
01750                             currentGlyphPlacementDelta = new Point(currentGlyphPlacementDelta.X +
      kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y + kerning.Glyph1Placement.Y);
```

```
01751                             currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
      kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
01752
01753                             nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
      kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
01754                             nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
      kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
01755                         }
01756                     }
01757
01758                     if (currentGlyphPlacementDelta.X != 0 || currentGlyphPlacementDelta.Y != 0 ||
      currentGlyphAdvanceDelta.X != 0 || currentGlyphAdvanceDelta.Y != 0)
01759                     {
01760                         if (currentRun.Length > 0)
01761                         {
01762                             tSpans.Add((currentRun.ToString(), currentKerning));
01763
01764                             tSpans.Add((text[i].ToString(), new Point(currentGlyphPlacementDelta.X *
      Font.FontSize / 1000, currentGlyphPlacementDelta.Y * Font.FontSize / 1000)));
01765
01766                             currentRun.Clear();
01767                             currentKerning = new Point((currentGlyphAdvanceDelta.X -
      currentGlyphPlacementDelta.X) * Font.FontSize / 1000, (currentGlyphAdvanceDelta.Y -
      currentGlyphPlacementDelta.Y) * Font.FontSize / 1000);
01768                         }
01769                         else
01770                         {
01771                             tSpans.Add((text[i].ToString(), new Point(currentGlyphPlacementDelta.X *
      Font.FontSize / 1000 + currentKerning.X, currentGlyphPlacementDelta.Y * Font.FontSize / 1000 +
      currentKerning.Y)));
01772
01773                             currentRun.Clear();
01774                             currentKerning = new Point((currentGlyphAdvanceDelta.X -
      currentGlyphPlacementDelta.X) * Font.FontSize / 1000, (currentGlyphAdvanceDelta.Y -
      currentGlyphPlacementDelta.Y) * Font.FontSize / 1000);
01775                         }
01776                     }
01777                     else
01778                     {
01779                         currentRun.Append(text[i]);
01780                     }
01781                 }
01782
01783                 if (currentRun.Length > 0)
01784                 {
01785                     tSpans.Add((currentRun.ToString(), currentKerning));
01786                 }
01787
01788                 double currX = x;
01789                 double currY = y;
01790
01791                 Font.DetailedFontMetrics fullMetrics = Font.MeasureTextAdvanced(text);
01792
01793                 if (TextBaseline == TextBaselines.Top)
01794                 {
01795                     if (Font.FontFamily.TrueTypeFile != null)
01796                     {
01797                         currY += fullMetrics.Top;
01798                     }
01799                 }
01800                 else if (TextBaseline == TextBaselines.Middle)
01801                 {
01802                     if (Font.FontFamily.TrueTypeFile != null)
01803                     {
01804                         currY += (fullMetrics.Top + fullMetrics.Bottom) * 0.5;
01805                     }
01806                 }
01807                 else if (TextBaseline == TextBaselines.Bottom)
01808                 {
01809                     if (Font.FontFamily.TrueTypeFile != null)
01810                     {
01811                         currY += fullMetrics.Bottom;
01812                     }
01813                 }
01814
01815                 TextBaseline = TextBaselines.Baseline;
01816
01817                 for (int i = 0; i < tSpans.Count; i++)
01818                 {
01819                     Font.DetailedFontMetrics metrics = Font.MeasureTextAdvanced(tSpans[i].Item1);
01820
01821                     if (i == 0)
01822                     {
01823                         FillSimpleText(tSpans[i].Item1, currX + tSpans[i].Item2.X, currY +
      tSpans[i].Item2.Y);
01824                     }
01825                     else
```

```
01826                     {
01827                         FillSimpleText(tSpans[i].Item1, currX + metrics.LeftSideBearing -
      fullMetrics.LeftSideBearing + tSpans[i].Item2.X, currY + tSpans[i].Item2.Y);
01828                     }
01829 
01830 
01831                     currX += metrics.AdvanceWidth + tSpans[i].Item2.X;
01832                     currY += tSpans[i].Item2.Y;
01833                 }
01834             }
01835         }
01836 
01837         public void FillSimpleText(string text, double x, double y)
01838         {
01839             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
01840 
01841             if (_textOption == AvaloniaContextInterpreter.TextOptions.NeverConvert || (_textOption ==
      AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary && Font.FontFamily.IsStandardFamily &&
      Font.FontFamily.FileName != "ZapfDingbats" && Font.FontFamily.FileName != "Symbol"))
01842             {
01843                 Typeface typeface = new Typeface(Avalonia.Media.FontFamily.Parse(FontFamily),
      (Font.FontFamily.IsOblique ?  FontStyle.Oblique :  Font.FontFamily.IsItalic ?  FontStyle.Italic :
      FontStyle.Normal), (Font.FontFamily.IsBold ?  FontWeight.Bold :  FontWeight.Regular));
01844 
01845                 Avalonia.Media.FormattedText txt = new Avalonia.Media.FormattedText(text,
      System.Globalization.CultureInfo.InvariantCulture, FlowDirection.LeftToRight, typeface, Font.FontSize,
      null);
01846 
01847                 double top = y;
01848                 double left = x;
01849 
01850                 double[,] currTransform = null;
01851                 double[,] deltaTransform = MatrixUtils.Identity;
01852 
01853                 if (Font.FontFamily.TrueTypeFile != null)
01854                 {
01855                     currTransform = MatrixUtils.Translate(_transform, x, y);
01856                 }
01857 
01858                 Font.DetailedFontMetrics metrics = Font.MeasureTextAdvanced(text);
01859 
01860                 if (text.StartsWith(" "))
01861                 {
01862                     var spaceMetrics = Font.MeasureTextAdvanced(" ");
01863 
01864                     for (int i = 0; i < text.Length; i++)
01865                     {
01866                         if (text[i] == ' ')
01867                         {
01868                             left -= spaceMetrics.AdvanceWidth;
01869                         }
01870                         else
01871                         {
01872                             break;
01873                         }
01874                     }
01875                 }
01876 
01877 
01878                 if (TextBaseline == TextBaselines.Top)
01879                 {
01880                     if (Font.FontFamily.TrueTypeFile != null)
01881                     {
01882                         if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
01883                         {
01884                             currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing, top + metrics.Top - Font.WinAscent);
01885                             deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing, top + metrics.Top - Font.WinAscent);
01886                         }
01887                         else
01888                         {
01889                             currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing, top + metrics.Top - Font.Ascent);
01890                             deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing, top + metrics.Top - Font.Ascent);
01891                         }
01892 
01893                     }
01894                 }
01895                 else if (TextBaseline == TextBaselines.Middle)
01896                 {
01897                     if (Font.FontFamily.TrueTypeFile != null)
01898                     {
01899                         if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
01900                         {
01901                             currTransform = MatrixUtils.Translate(_transform, left -
```

```
      metrics.LeftSideBearing, top + metrics.Top / 2 + metrics.Bottom / 2 - Font.WinAscent);
01902                             deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing, top + metrics.Top / 2 + metrics.Bottom / 2 - Font.WinAscent);
01903                         }
01904                         else
01905                         {
01906                             currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing, top + metrics.Top / 2 + metrics.Bottom / 2 - Font.Ascent);
01907                             deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing, top + metrics.Top / 2 + metrics.Bottom / 2 - Font.Ascent);
01908                         }
01909                     }
01910                 }
01911                 else if (TextBaseline == TextBaselines.Baseline)
01912                 {
01913                     if (Font.FontFamily.TrueTypeFile != null)
01914                     {
01915                         if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
01916                         {
01917                             currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing, top - Font.WinAscent);
01918                             deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing, top - Font.YMax);
01919                         }
01920                         else
01921                         {
01922                             currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing, top - Font.Ascent);
01923                             deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing, top - Font.YMax);
01924                         }
01925                     }
01926                 }
01927                 else if (TextBaseline == TextBaselines.Bottom)
01928                 {
01929                     if (Font.FontFamily.TrueTypeFile != null)
01930                     {
01931                         if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
01932                         {
01933                             currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing, top - Font.WinAscent + metrics.Bottom);
01934                             deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing, top - Font.WinAscent + metrics.Bottom);
01935                         }
01936                         else
01937                         {
01938                             currTransform = MatrixUtils.Translate(_transform, left -
      metrics.LeftSideBearing, top - Font.Ascent + metrics.Bottom);
01939                             deltaTransform = MatrixUtils.Translate(deltaTransform, left -
      metrics.LeftSideBearing, top - Font.Ascent + metrics.Bottom);
01940                         }
01941                     }
01942                 }
01943
01944                 Avalonia.Media.Brush fill = null;
01945
01946                 if (this.FillStyle is SolidColourBrush solid)
01947                 {
01948                     fill = new SolidColorBrush(Color.FromArgb(FillAlpha, (byte)(solid.R * 255),
      (byte)(solid.G * 255), (byte)(solid.B * 255)));
01949                 }
01950                 else if (this.FillStyle is LinearGradientBrush linearGradient)
01951                 {
01952                     fill = linearGradient.ToLinearGradientBrush(deltaTransform);
01953                 }
01954                 else if (this.FillStyle is RadialGradientBrush radialGradient)
01955                 {
01956                     fill = radialGradient.ToRadialGradientBrush(metrics.Width +
      metrics.LeftSideBearing + metrics.RightSideBearing, deltaTransform);
01957                 }
01958
01959                 txt.SetForegroundBrush(fill);
01960
01961                 TextRunProperties runProperties = new GenericTextRunProperties(typeface,
      Font.FontSize);
01962
01963                 TextLayout lay = new TextLayout(new SimpleTextSource(text, runProperties), new
      GenericTextParagraphProperties(runProperties));
01964
01965                 RenderAction act = RenderAction.TextAction(txt, lay, fill,
      currTransform.ToAvaloniaMatrix(), _clippingPath?.Clone(), Tag);
01966
01967                 if (!string.IsNullOrEmpty(Tag))
01968                 {
01969                     if (TaggedActions.ContainsKey(Tag))
01970                     {
01971                         IEnumerable<RenderAction> actions = TaggedActions[Tag](act);
```

```
01972
01973                       foreach (RenderAction action in actions)
01974                       {
01975                           RenderActions.Add(action);
01976                       }
01977
01978                       if (removeTaggedActions)
01979                       {
01980                           TaggedActions.Remove(Tag);
01981                       }
01982                   }
01983                   else
01984                   {
01985                       RenderActions.Add(act);
01986                   }
01987               }
01988               else if (TaggedActions.ContainsKey(""))
01989               {
01990                   IEnumerable<RenderAction> actions = TaggedActions[""](act);
01991
01992                   foreach (RenderAction action in actions)
01993                   {
01994                       RenderActions.Add(action);
01995                   }
01996               }
01997               else
01998               {
01999                   RenderActions.Add(act);
02000               }
02001           }
02002           else
02003           {
02004               PathText(text, x, y);
02005               Fill(FillRule.NonZeroWinding);
02006           }
02007       }
02008
02009       public Brush StrokeStyle { get; private set; } = Colour.FromRgb(0, 0, 0);
02010       private byte StrokeAlpha = 255;
02011
02012       public Brush FillStyle { get; private set; } = Colour.FromRgb(0, 0, 0);
02013       private byte FillAlpha = 255;
02014
02015       public void SetFillStyle((int r, int g, int b, double a) style)
02016       {
02017           FillStyle = Colour.FromRgba(style.r, style.g, style.b, (int)(style.a * 255));
02018           FillAlpha = (byte)(style.a * 255);
02019       }
02020
02021       public void SetFillStyle(Brush style)
02022       {
02023           FillStyle = style;
02024
02025           if (style is SolidColourBrush solid)
02026           {
02027               FillAlpha = (byte)(solid.A * 255);
02028           }
02029           else
02030           {
02031               FillAlpha = 255;
02032           }
02033       }
02034
02035       public void SetStrokeStyle((int r, int g, int b, double a) style)
02036       {
02037           StrokeStyle = Colour.FromRgba(style.r, style.g, style.b, (int)(style.a * 255));
02038           StrokeAlpha = (byte)(style.a * 255);
02039       }
02040
02041       public void SetStrokeStyle(Brush style)
02042       {
02043           StrokeStyle = style;
02044
02045           if (style is SolidColourBrush solid)
02046           {
02047               StrokeAlpha = (byte)(solid.A * 255);
02048           }
02049           else
02050           {
02051               StrokeAlpha = 255;
02052           }
02053       }
02054
02055       private LineDash LineDash;
02056
02057       public void SetLineDash(LineDash dash)
02058       {
```

```
02059             LineDash = new LineDash(dash.DashArray, dash.Phase);
02060         }
02061
02062         public void Rotate(double angle)
02063         {
02064             Utils.CoerceNaNAndInfinityToZero(ref angle);
02065
02066             _transform = MatrixUtils.Rotate(_transform, angle);
02067
02068             currentPath = new PathGeometry();
02069             currentFigure = new PathFigure() { IsClosed = false };
02070             figureInitialised = false;
02071         }
02072
02073         public void Transform(double a, double b, double c, double d, double e, double f)
02074         {
02075             Utils.CoerceNaNAndInfinityToZero(ref a, ref b, ref c, ref d, ref e, ref f);
02076
02077             double[,] transfMatrix = new double[3, 3] { { a, c, e }, { b, d, f }, { 0, 0, 1 } };
02078             _transform = MatrixUtils.Multiply(_transform, transfMatrix);
02079
02080             currentPath = new PathGeometry();
02081             currentFigure = new PathFigure() { IsClosed = false };
02082             figureInitialised = false;
02083         }
02084
02085         public void Scale(double x, double y)
02086         {
02087             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
02088
02089             _transform = MatrixUtils.Scale(_transform, x, y);
02090
02091             currentPath = new PathGeometry();
02092             currentFigure = new PathFigure() { IsClosed = false };
02093             figureInitialised = false;
02094         }
02095
02096         private double[,] _transform;
02097
02098         private readonly Stack<double[,]> states;
02099
02100         public void Save()
02101         {
02102             states.Push((double[,])_transform.Clone());
02103             clippingPaths.Push(_clippingPath?.Clone());
02104         }
02105
02106         public void Restore()
02107         {
02108             _transform = states.Pop();
02109             _clippingPath = clippingPaths.Pop();
02110         }
02111
02112         public double LineWidth { get; set; }
02113         public LineCaps LineCap { get; set; }
02114         public LineJoins LineJoin { get; set; }
02115
02116         private string FontFamily;
02117         private Font _Font;
02118
02119         public Font Font
02120         {
02121             get
02122             {
02123                 return _Font;
02124             }
02125
02126             set
02127             {
02128                 _Font = value;
02129
02130                 if (!Font.FontFamily.IsStandardFamily)
02131                 {
02132                     if (Font.FontFamily is ResourceFontFamily fam)
02133                     {
02134                         FontFamily = fam.ResourceName;
02135                     }
02136                     else
02137                     {
02138                         if (Font.FontFamily.TrueTypeFile != null)
02139                         {
02140                             FontFamily = Font.FontFamily.TrueTypeFile.GetFontFamilyName();
02141                         }
02142                         else
02143                         {
02144                             FontFamily = Font.FontFamily.FileName;
02145                         }
```

```
02146                     }
02147                 }
02148                 else
02149                 {
02150                     FontFamily = "resm:VectSharp.StandardFonts.?assembly=VectSharp#" +
      Font.FontFamily.TrueTypeFile.GetFontFamilyName();
02151                 }
02152             }
02153         }
02154 
02155         private PathGeometry currentPath;
02156         private PathFigure currentFigure;
02157 
02158         private bool figureInitialised = false;
02159 
02160         public void MoveTo(double x, double y)
02161         {
02162             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
02163 
02164             if (figureInitialised)
02165             {
02166                 currentPath.Figures.Add(currentFigure);
02167             }
02168 
02169             currentFigure = new PathFigure() { StartPoint = new Avalonia.Point(x, y), IsClosed = false
      };
02170             figureInitialised = true;
02171         }
02172 
02173         public void LineTo(double x, double y)
02174         {
02175             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
02176 
02177             if (figureInitialised)
02178             {
02179                 currentFigure.Segments.Add(new Avalonia.Media.LineSegment() { Point = new
      Avalonia.Point(x, y) });
02180             }
02181             else
02182             {
02183                 currentFigure = new PathFigure() { StartPoint = new Avalonia.Point(x, y), IsClosed =
      false };
02184                 figureInitialised = true;
02185             }
02186         }
02187 
02188         public void Rectangle(double x0, double y0, double width, double height)
02189         {
02190             Utils.CoerceNaNAndInfinityToZero(ref x0, ref y0, ref width, ref height);
02191 
02192             if (currentFigure != null && figureInitialised)
02193             {
02194                 currentPath.Figures.Add(currentFigure);
02195             }
02196 
02197             currentFigure = new PathFigure() { StartPoint = new Avalonia.Point(x0, y0), IsClosed =
      false };
02198             currentFigure.Segments.Add(new Avalonia.Media.LineSegment() { Point = new
      Avalonia.Point(x0 + width, y0) });
02199             currentFigure.Segments.Add(new Avalonia.Media.LineSegment() { Point = new
      Avalonia.Point(x0 + width, y0 + height) });
02200             currentFigure.Segments.Add(new Avalonia.Media.LineSegment() { Point = new
      Avalonia.Point(x0, y0 + height) });
02201             currentFigure.IsClosed = true;
02202 
02203             currentPath.Figures.Add(currentFigure);
02204             figureInitialised = false;
02205         }
02206 
02207         public void CubicBezierTo(double p1X, double p1Y, double p2X, double p2Y, double p3X, double
      p3Y)
02208         {
02209             Utils.CoerceNaNAndInfinityToZero(ref p1X, ref p1Y, ref p2X, ref p2Y, ref p3X, ref p3Y);
02210 
02211             if (figureInitialised)
02212             {
02213                 currentFigure.Segments.Add(new Avalonia.Media.BezierSegment() { Point1 = new
      Avalonia.Point(p1X, p1Y), Point2 = new Avalonia.Point(p2X, p2Y), Point3 = new Avalonia.Point(p3X, p3Y)
      });
02214             }
02215             else
02216             {
02217                 currentFigure = new PathFigure() { StartPoint = new Avalonia.Point(p1X, p1Y), IsClosed
      = false };
02218                 figureInitialised = true;
02219             }
02220         }
```

```
02221
02222         public void Close()
02223         {
02224             currentFigure.IsClosed = true;
02225             currentPath.Figures.Add(currentFigure);
02226             figureInitialised = false;
02227         }
02228
02229         public void Stroke()
02230         {
02231             if (figureInitialised)
02232             {
02233                 currentFigure.IsClosed = false;
02234                 currentPath.Figures.Add(currentFigure);
02235             }
02236
02237             Avalonia.Media.Brush stroke = null;
02238
02239             if (this.StrokeStyle is SolidColourBrush solid)
02240             {
02241                 stroke = new SolidColorBrush(Color.FromArgb(StrokeAlpha, (byte)(solid.R * 255),
      (byte)(solid.G * 255), (byte)(solid.B * 255)));
02242             }
02243             else if (this.StrokeStyle is LinearGradientBrush linearGradient)
02244             {
02245                 stroke = linearGradient.ToLinearGradientBrush();
02246             }
02247             else if (this.StrokeStyle is RadialGradientBrush radialGradient)
02248             {
02249                 stroke = radialGradient.ToRadialGradientBrush(currentPath.Bounds.Width);
02250             }
02251
02252             Pen pen = new Pen(stroke,
02253                     LineWidth,
02254                     new DashStyle(LineDash.DashArray.Select(x => (x + (LineCap == LineCaps.Butt ?  0 :
      LineWidth)) / LineWidth), LineDash.Phase / LineWidth));
02255
02256             switch (LineCap)
02257             {
02258                 case LineCaps.Butt:
02259                     pen.LineCap = PenLineCap.Flat;
02260                     break;
02261                 case LineCaps.Round:
02262                     pen.LineCap = PenLineCap.Round;
02263                     break;
02264                 case LineCaps.Square:
02265                     pen.LineCap = PenLineCap.Square;
02266                     break;
02267             }
02268
02269             switch (LineJoin)
02270             {
02271                 case LineJoins.Bevel:
02272                     pen.LineJoin = PenLineJoin.Bevel;
02273                     break;
02274                 case LineJoins.Round:
02275                     pen.LineJoin = PenLineJoin.Round;
02276                     break;
02277                 case LineJoins.Miter:
02278                     pen.LineJoin = PenLineJoin.Miter;
02279                     break;
02280             }
02281
02282             RenderAction act = RenderAction.PathAction(currentPath, pen, null,
      _transform.ToAvaloniaMatrix(), _clippingPath?.Clone(), Tag);
02283
02284             if (!string.IsNullOrEmpty(Tag))
02285             {
02286                 if (TaggedActions.ContainsKey(Tag))
02287                 {
02288                     IEnumerable<RenderAction> actions = TaggedActions[Tag](act);
02289
02290                     foreach (RenderAction action in actions)
02291                     {
02292                         RenderActions.Add(action);
02293                     }
02294
02295                     if (removeTaggedActions)
02296                     {
02297                         TaggedActions.Remove(Tag);
02298                     }
02299                 }
02300                 else
02301                 {
02302                     RenderActions.Add(act);
02303                 }
02304             }
```

```
02305             else if (TaggedActions.ContainsKey(""))
02306             {
02307                 IEnumerable<RenderAction> actions = TaggedActions[""](act);
02308 
02309                 foreach (RenderAction action in actions)
02310                 {
02311                     RenderActions.Add(action);
02312                 }
02313             }
02314             else
02315             {
02316                 RenderActions.Add(act);
02317             }
02318 
02319             currentPath = new PathGeometry();
02320             currentFigure = new PathFigure() { IsClosed = false };
02321             figureInitialised = false;
02322         }
02323 
02324         public void Fill(FillRule fillRule)
02325         {
02326             if (figureInitialised)
02327             {
02328                 currentPath.Figures.Add(currentFigure);
02329             }
02330 
02331             Avalonia.Media.Brush fill = null;
02332 
02333             if (this.FillStyle is SolidColourBrush solid)
02334             {
02335                 fill = new SolidColorBrush(Color.FromArgb(FillAlpha, (byte)(solid.R * 255),
    (byte)(solid.G * 255), (byte)(solid.B * 255)));
02336             }
02337             else if (this.FillStyle is LinearGradientBrush linearGradient)
02338             {
02339                 fill = linearGradient.ToLinearGradientBrush();
02340             }
02341             else if (this.FillStyle is RadialGradientBrush radialGradient)
02342             {
02343                 fill = radialGradient.ToRadialGradientBrush(currentPath.Bounds.Width);
02344             }
02345 
02346             switch (fillRule)
02347             {
02348                 case FillRule.NonZeroWinding:
02349                     currentPath.FillRule = Avalonia.Media.FillRule.NonZero;
02350                     break;
02351 
02352                 case FillRule.EvenOdd:
02353                     currentPath.FillRule = Avalonia.Media.FillRule.EvenOdd;
02354                     break;
02355             }
02356 
02357             RenderAction act = RenderAction.PathAction(currentPath, null, fill,
    _transform.ToAvaloniaMatrix(), _clippingPath?.Clone(), Tag);
02358 
02359             if (!string.IsNullOrEmpty(Tag))
02360             {
02361                 if (TaggedActions.ContainsKey(Tag))
02362                 {
02363                     IEnumerable<RenderAction> actions = TaggedActions[Tag](act);
02364 
02365                     foreach (RenderAction action in actions)
02366                     {
02367                         RenderActions.Add(action);
02368                     }
02369 
02370                     if (removeTaggedActions)
02371                     {
02372                         TaggedActions.Remove(Tag);
02373                     }
02374                 }
02375                 else
02376                 {
02377                     RenderActions.Add(act);
02378                 }
02379             }
02380             else if (TaggedActions.ContainsKey(""))
02381             {
02382                 IEnumerable<RenderAction> actions = TaggedActions[""](act);
02383 
02384                 foreach (RenderAction action in actions)
02385                 {
02386                     RenderActions.Add(action);
02387                 }
02388             }
02389             else
```

```
02390             {
02391                 RenderActions.Add(act);
02392             }
02393 
02394             currentPath = new PathGeometry();
02395             currentFigure = new PathFigure() { IsClosed = false };
02396             figureInitialised = false;
02397         }
02398 
02399         private static void TransformGeometry(PathGeometry geo, double[,] transf)
02400         {
02401             for (int i = 0; i < geo.Figures.Count; i++)
02402             {
02403                 double[] tP = MatrixUtils.Multiply(transf, new double[] { geo.Figures[i].StartPoint.X,
      geo.Figures[i].StartPoint.Y });
02404                 geo.Figures[i].StartPoint = new Avalonia.Point(tP[0], tP[1]);
02405 
02406                 for (int j = 0; j < geo.Figures[i].Segments.Count; j++)
02407                 {
02408                     if (geo.Figures[i].Segments[j] is LineSegment lS)
02409                     {
02410                         tP = MatrixUtils.Multiply(transf, new double[] { lS.Point.X, lS.Point.Y });
02411                         lS.Point = new Avalonia.Point(tP[0], tP[1]);
02412                     }
02413                     else if (geo.Figures[i].Segments[j] is BezierSegment bS)
02414                     {
02415                         tP = MatrixUtils.Multiply(transf, new double[] { bS.Point1.X, bS.Point1.Y });
02416                         double[] tP2 = MatrixUtils.Multiply(transf, new double[] { bS.Point2.X,
      bS.Point2.Y });
02417                         double[] tP3 = MatrixUtils.Multiply(transf, new double[] { bS.Point3.X,
      bS.Point3.Y });
02418 
02419                         bS.Point1 = new Avalonia.Point(tP[0], tP[1]);
02420                         bS.Point2 = new Avalonia.Point(tP2[0], tP2[1]);
02421                         bS.Point3 = new Avalonia.Point(tP3[0], tP3[1]);
02422                     }
02423                 }
02424             }
02425         }
02426 
02427         public void SetClippingPath()
02428         {
02429             if (figureInitialised)
02430             {
02431                 currentPath.Figures.Add(currentFigure);
02432             }
02433 
02434             TransformGeometry(currentPath, _transform);
02435 
02436             if (_clippingPath == null)
02437             {
02438                 _clippingPath = currentPath;
02439             }
02440             else
02441             {
02442                 _clippingPath = new CombinedGeometry(GeometryCombineMode.Intersect, _clippingPath,
      currentPath);
02443             }
02444 
02445             currentPath = new PathGeometry();
02446             currentFigure = new PathFigure() { IsClosed = false };
02447             figureInitialised = false;
02448         }
02449 
02450         public void DrawRasterImage(int sourceX, int sourceY, int sourceWidth, int sourceHeight,
      double destinationX, double destinationY, double destinationWidth, double destinationHeight,
      RasterImage image)
02451         {
02452             Utils.CoerceNaNAndInfinityToZero(ref destinationX, ref destinationY, ref destinationWidth,
      ref destinationHeight);
02453 
02454             if (!this.Images.ContainsKey(image.Id))
02455             {
02456                 Bitmap bmp = new Bitmap(image.PNGStream);
02457                 this.Images.Add(image.Id, (bmp, image.Interpolate));
02458             }
02459 
02460             RenderAction act = RenderAction.ImageAction(image.Id, new Avalonia.Rect(sourceX, sourceY,
      sourceWidth, sourceHeight), new Avalonia.Rect(destinationX, destinationY, destinationWidth,
      destinationHeight), _transform.ToAvaloniaMatrix(), _clippingPath?.Clone(), Tag);
02461 
02462             if (!string.IsNullOrEmpty(Tag))
02463             {
02464                 if (TaggedActions.ContainsKey(Tag))
02465                 {
02466                     IEnumerable<RenderAction> actions = TaggedActions[Tag](act);
02467 
```

```
02468                     foreach (RenderAction action in actions)
02469                     {
02470                         RenderActions.Add(action);
02471                     }
02472 
02473                     if (removeTaggedActions)
02474                     {
02475                         TaggedActions.Remove(Tag);
02476                     }
02477                 }
02478                 else
02479                 {
02480                     RenderActions.Add(act);
02481                 }
02482             }
02483             else if (TaggedActions.ContainsKey(""))
02484             {
02485                 IEnumerable<RenderAction> actions = TaggedActions[""](act);
02486 
02487                 foreach (RenderAction action in actions)
02488                 {
02489                     RenderActions.Add(action);
02490                 }
02491             }
02492             else
02493             {
02494                 RenderActions.Add(act);
02495             }
02496         }
02497 
02498         public void DrawFilteredGraphics(Graphics graphics, IFilter filter)
02499         {
02500             if (this._filterOption.Operation == FilterOption.FilterOperations.RasteriseAllWithSkia)
02501             {
02502                 double scale = this._filterOption.RasterisationResolution;
02503 
02504                 Rectangle bounds = graphics.GetBounds();
02505 
02506                 bounds = new Rectangle(bounds.Location.X - filter.TopLeftMargin.X, bounds.Location.Y -
      filter.TopLeftMargin.Y, bounds.Size.Width + filter.TopLeftMargin.X + filter.BottomRightMargin.X,
      bounds.Size.Height + filter.TopLeftMargin.Y + filter.BottomRightMargin.Y);
02507 
02508                 if (bounds.Size.Width > 0 && bounds.Size.Height > 0)
02509                 {
02510                     if (!this._filterOption.RasterisationResolutionRelative)
02511                     {
02512                         scale = scale / Math.Min(bounds.Size.Width, bounds.Size.Height);
02513                     }
02514 
02515                     RasterImage rasterised = SKRenderContextInterpreter.Rasterise(graphics, bounds,
      scale, true);
02516                     RasterImage filtered = null;
02517 
02518                     if (filter is IFilterWithRasterisableParameter filterWithRastParam)
02519                     {
02520                         filterWithRastParam.RasteriseParameter(SKRenderContextInterpreter.Rasterise,
      scale);
02521                     }
02522 
02523                     if (filter is ILocationInvariantFilter locInvFilter)
02524                     {
02525                         filtered = locInvFilter.Filter(rasterised, scale);
02526                     }
02527                     else if (filter is IFilterWithLocation filterWithLoc)
02528                     {
02529                         filtered = filterWithLoc.Filter(rasterised, bounds, scale);
02530                     }
02531 
02532                     if (filtered != null)
02533                     {
02534                         rasterised.Dispose();
02535 
02536                         DrawRasterImage(0, 0, filtered.Width, filtered.Height, bounds.Location.X,
      bounds.Location.Y, bounds.Size.Width, bounds.Size.Height, filtered);
02537                     }
02538                 }
02539             }
02540             else if (this._filterOption.Operation ==
      FilterOption.FilterOperations.RasteriseAllWithVectSharp)
02541             {
02542                 double scale = this._filterOption.RasterisationResolution;
02543 
02544                 Rectangle bounds = graphics.GetBounds();
02545 
02546                 bounds = new Rectangle(bounds.Location.X - filter.TopLeftMargin.X, bounds.Location.Y -
      filter.TopLeftMargin.Y, bounds.Size.Width + filter.TopLeftMargin.X + filter.BottomRightMargin.X,
      bounds.Size.Height + filter.TopLeftMargin.Y + filter.BottomRightMargin.Y);
```

```
02547
02548                 if (bounds.Size.Width > 0 && bounds.Size.Height > 0)
02549                 {
02550                     if (!this._filterOption.RasterisationResolutionRelative)
02551                     {
02552                         scale = scale / Math.Min(bounds.Size.Width, bounds.Size.Height);
02553                     }
02554
02555                     if (graphics.TryRasterise(bounds, scale, true, out RasterImage rasterised))
02556                     {
02557                         RasterImage filtered = null;
02558
02559                         if (filter is IFilterWithRasterisableParameter filterWithRastParam)
02560                         {
02561
      filterWithRastParam.RasteriseParameter(SKRenderContextInterpreter.Rasterise, scale);
02562                         }
02563
02564                         if (filter is ILocationInvariantFilter locInvFilter)
02565                         {
02566                             filtered = locInvFilter.Filter(rasterised, scale);
02567                         }
02568                         else if (filter is IFilterWithLocation filterWithLoc)
02569                         {
02570                             filtered = filterWithLoc.Filter(rasterised, bounds, scale);
02571                         }
02572
02573                         if (filtered != null)
02574                         {
02575                             rasterised.Dispose();
02576
02577                             DrawRasterImage(0, 0, filtered.Width, filtered.Height, bounds.Location.X,
      bounds.Location.Y, bounds.Size.Width, bounds.Size.Height, filtered);
02578                         }
02579                     }
02580                     else
02581                     {
02582                         throw new NotImplementedException(@"The filter could not be rasterised!  You
      can avoid this error by doing one of the following:
02583 • Add a reference to VectSharp.Raster or VectSharp.Raster.ImageSharp (you may also need to add a using
      directive somewhere to force the assembly to be loaded).
02584 • Provide your own implementation of Graphics.RasterisationMethod.
02585 • Set the FilterOption.Operation to ""RasteriseAllWithSkia"", ""IgnoreAll"" or ""SkipAll"".");
02586                     }
02587                 }
02588             }
02589             else if (this._filterOption.Operation == FilterOption.FilterOperations.IgnoreAll)
02590             {
02591                 graphics.CopyToIGraphicsContext(this);
02592             }
02593             else
02594             {
02595
02596             }
02597         }
02598     }
02599
02600
02601 /// <summary>
02602 /// Contains methods to render a <see cref="Page"/> to an <see cref="Avalonia.Controls.Canvas"/>.
02603 /// </summary>
02604     public static class AvaloniaContextInterpreter
02605     {
02606 /// <summary>
02607 /// Defines whether text items should be converted into paths when drawing.
02608 /// </summary>
02609         public enum TextOptions
02610         {
02611 /// <summary>
02612 /// Converts all text items into paths.
02613 /// </summary>
02614             AlwaysConvert,
02615
02616 /// <summary>
02617 /// Converts all text items into paths, with the exception of those that use a standard font.
02618 /// </summary>
02619             ConvertIfNecessary,
02620
02621 /// <summary>
02622 /// Does not convert any text items into paths.
02623 /// </summary>
02624             NeverConvert
02625         }
02626
02627 /// <summary>
02628 /// Render a <see cref="Page"/> to an <see cref="Avalonia.Controls.Canvas"/>.
02629 /// </summary>
```

```
02630 /// <param name="page">The <see cref="Page"/> to render.</param>
02631 /// <param name="textOption">Defines whether text items should be converted into paths when
      drawing.</param>
02632 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
02633 /// <returns>An <see cref="Avalonia.Controls.Canvas"/> containing the rendered graphics
      objects.</returns>
02634         public static Avalonia.Controls.Canvas PaintToCanvas(this Page page, TextOptions textOption =
      TextOptions.ConvertIfNecessary, FilterOption filterOption = default)
02635         {
02636             filterOption = filterOption ??  FilterOption.Default;
02637 
02638             AvaloniaContext ctx = new AvaloniaContext(page.Width, page.Height, true, textOption,
      filterOption);
02639             page.Graphics.CopyToIGraphicsContext(ctx);
02640             ctx.ControlItem.Background = new SolidColorBrush(Color.FromArgb((byte)(page.Background.A *
      255), (byte)(page.Background.R * 255), (byte)(page.Background.G * 255), (byte)(page.Background.B *
      255)));
02641             return ctx.ControlItem;
02642         }
02643 
02644 /// <summary>
02645 /// Render a <see cref="Page"/> to an <see cref="Avalonia.Controls.Control"/>.
02646 /// </summary>
02647 /// <param name="page">The <see cref="Page"/> to render.</param>
02648 /// <param name="graphicsAsControls">If this is true, each graphics object (e.g.  paths, text...)  is
      rendered as a separate <see cref="Avalonia.Controls.Control"/>.  Otherwise, they are directly rendered
      onto the drawing context (which is faster, but does not allow interactivity).</param>
02649 /// <param name="textOption">Defines whether text items should be converted into paths when
      drawing.</param>
02650 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
02651 /// <returns>An <see cref="Avalonia.Controls.Control"/> containing the rendered graphics
      objects.</returns>
02652         public static Avalonia.Controls.Control PaintToCanvas(this Page page, bool graphicsAsControls,
      TextOptions textOption = TextOptions.ConvertIfNecessary, FilterOption filterOption = default)
02653         {
02654             filterOption = filterOption ??  FilterOption.Default;
02655 
02656             if (graphicsAsControls)
02657             {
02658                 Avalonia.Controls.Canvas tbr = page.PaintToCanvas(textOption, filterOption);
02659                 tbr.Background = new SolidColorBrush(Color.FromArgb((byte)(page.Background.A * 255),
      (byte)(page.Background.R * 255), (byte)(page.Background.G * 255), (byte)(page.Background.B * 255)));
02660                 return tbr;
02661             }
02662             else
02663             {
02664                 return new RenderCanvas(page.Graphics, page.Background, page.Width, page.Height, new
      Dictionary<string, Delegate>(), true, textOption, filterOption);
02665             }
02666         }
02667 
02668 /// <summary>
02669 /// Render a <see cref="Page"/> to an <see cref="Avalonia.Controls.Control"/>.
02670 /// </summary>
02671 /// <param name="page">The <see cref="Page"/> to render.</param>
02672 /// <param name="graphicsAsControls">If this is true, each graphics object (e.g.  paths, text...)  is
      rendered as a separate <see cref="Avalonia.Controls.Control"/>.  Otherwise, they are directly rendered
      onto the drawing context (which is faster, but does not allow interactivity).</param>
02673 /// <param name="taggedActions">A <see cref="Dictionary{String, Delegate}"/> containing the <see
      cref="Action"/>s that will be performed on items with the corresponding tag.
02674 /// If <paramref name="graphicsAsControls"/> is true, the delegates should be voids that accept one
      parameter of type <see cref="TextBlock"/> or <see cref="Path"/> (depending on the tagged item),
      otherwise, they should accept one parameter of type <see cref="RenderAction"/> and return an <see
      cref="IEnumerable{RenderAction}"/> of the actions that will actually be performed.</param>
02675 /// <param name="removeTaggedActionsAfterExecution">Whether the <see cref="Action"/>s should be
      removed from <paramref name="taggedActions"/> after their execution.  Set to false if the same <see
      cref="Action"/> should be performed on multiple items with the same tag.</param>
02676 /// <param name="textOption">Defines whether text items should be converted into paths when
      drawing.</param>
02677 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
02678 /// <returns>An <see cref="Avalonia.Controls.Control"/> containing the rendered graphics
      objects.</returns>
02679         public static Avalonia.Controls.Control PaintToCanvas(this Page page, bool graphicsAsControls,
      Dictionary<string, Delegate> taggedActions, bool removeTaggedActionsAfterExecution = true, TextOptions
      textOption = TextOptions.ConvertIfNecessary, FilterOption filterOption = default)
02680         {
02681             filterOption = filterOption ??  FilterOption.Default;
02682 
02683             if (graphicsAsControls)
02684             {
02685                 Avalonia.Controls.Canvas tbr = page.PaintToCanvas(taggedActions,
      removeTaggedActionsAfterExecution, textOption, filterOption);
02686                 tbr.Background = new SolidColorBrush(Color.FromArgb((byte)(page.Background.A * 255),
      (byte)(page.Background.R * 255), (byte)(page.Background.G * 255), (byte)(page.Background.B * 255)));
```

```
02687                 return tbr;
02688             }
02689             else
02690             {
02691                 return new RenderCanvas(page.Graphics, page.Background, page.Width, page.Height,
      taggedActions, removeTaggedActionsAfterExecution, textOption, filterOption);
02692             }
02693         }
02694 
02695 /// <summary>
02696 /// Render a <see cref="Page"/> to an <see cref="Avalonia.Controls.Control"/>.
02697 /// </summary>
02698 /// <param name="page">The <see cref="Page"/> to render.</param>
02699 /// <param name="taggedActions">A <see cref="Dictionary{String, Delegate}"/> containing the <see
      cref="Action"/>s that will be performed on items with the corresponding tag.
02700 /// The delegates should accept one parameter of type <see cref="TextBlock"/> or <see cref="Path"/>
      (depending on the tagged item).</param>
02701 /// <param name="removeTaggedActionsAfterExecution">Whether the <see cref="Action"/>s should be
      removed from <paramref name="taggedActions"/> after their execution.  Set to false if the same <see
      cref="Action"/> should be performed on multiple items with the same tag.</param>
02702 /// <param name="textOption">Defines whether text items should be converted into paths when
      drawing.</param>
02703 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
02704 /// <returns>An <see cref="Avalonia.Controls.Control"/> containing the rendered graphics
      objects.</returns>
02705         public static Avalonia.Controls.Canvas PaintToCanvas(this Page page, Dictionary<string,
      Delegate> taggedActions, bool removeTaggedActionsAfterExecution = true, TextOptions textOption =
      TextOptions.ConvertIfNecessary, FilterOption filterOption = default)
02706         {
02707             filterOption = filterOption ??  FilterOption.Default;
02708 
02709             AvaloniaContext ctx = new AvaloniaContext(page.Width, page.Height,
      removeTaggedActionsAfterExecution, textOption, filterOption)
02710             {
02711                 TaggedActions = taggedActions
02712             };
02713             page.Graphics.CopyToIGraphicsContext(ctx);
02714             ctx.ControlItem.Background = new SolidColorBrush(Color.FromArgb((byte)(page.Background.A *
      255), (byte)(page.Background.R * 255), (byte)(page.Background.G * 255), (byte)(page.Background.B *
      255)));
02715             return ctx.ControlItem;
02716         }
02717 
02718 /// <summary>
02719 /// Render an <see cref="Animation"/> to an <see cref="Avalonia.Controls.Control"/>.
02720 /// </summary>
02721 /// <param name="animation">The <see cref="Animation"/> to render.</param>
02722 /// <param name="frameRate">The target frame rate of the animation, in frames-per-second
      (fps).</param>
02723 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the
      animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note
      that this does not affect the frame rate of the animation.</param>
02724 /// <param name="textOption">Defines whether text items should be converted into paths when
      drawing.</param>
02725 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
02726 /// <returns>An <see cref="Avalonia.Controls.Control"/> containing the rendered animation.</returns>
02727         public static AnimatedCanvas PaintToAnimatedCanvas(this Animation animation, double frameRate
      = 60, double durationScaling = 1, TextOptions textOption = TextOptions.ConvertIfNecessary,
      FilterOption filterOption = default)
02728         {
02729             filterOption = filterOption ??  FilterOption.Default;
02730 
02731             return new AnimatedCanvas(animation, durationScaling, frameRate, textOption,
      filterOption);
02732         }
02733 
02734         internal static Avalonia.Media.LinearGradientBrush ToLinearGradientBrush(this
      LinearGradientBrush brush, double[,] transformMatrix = null)
02735         {
02736             Point start = brush.StartPoint;
02737             Point end = brush.EndPoint;
02738 
02739             if (transformMatrix != null)
02740             {
02741                 double[,] inverse = MatrixUtils.Invert(transformMatrix);
02742 
02743                 double[] startVec = MatrixUtils.Multiply(inverse, new double[] { start.X, start.Y });
02744                 double[] endVec = MatrixUtils.Multiply(inverse, new double[] { end.X, end.Y });
02745 
02746                 start = new Point(startVec[0], startVec[1]);
02747                 end = new Point(endVec[0], endVec[1]);
02748             }
02749 
02750             Avalonia.Media.LinearGradientBrush tbr = new Avalonia.Media.LinearGradientBrush()
02751             {
```

```
02752                 SpreadMethod = GradientSpreadMethod.Pad,
02753                 StartPoint = new Avalonia.RelativePoint(start.X, start.Y,
    Avalonia.RelativeUnit.Absolute),
02754                 EndPoint = new Avalonia.RelativePoint(end.X, end.Y, Avalonia.RelativeUnit.Absolute)
02755             };
02756
02757             Avalonia.Media.GradientStops stops = new Avalonia.Media.GradientStops();
02758             stops.AddRange(from el in brush.GradientStops select new
    Avalonia.Media.GradientStop(Color.FromArgb((byte)(el.Colour.A * 255), (byte)(el.Colour.R * 255),
    (byte)(el.Colour.G * 255), (byte)(el.Colour.B * 255)), el.Offset));
02759
02760             tbr.GradientStops = stops;
02761
02762             return tbr;
02763         }
02764
02765         internal static Avalonia.Media.RadialGradientBrush ToRadialGradientBrush(this
    RadialGradientBrush brush, double objectWidth, double[,] transformMatrix = null)
02766         {
02767             Point focus = brush.FocalPoint;
02768             Point centre = brush.Centre;
02769
02770             if (transformMatrix != null)
02771             {
02772                 double[,] inverse = MatrixUtils.Invert(transformMatrix);
02773
02774                 double[] focusVec = MatrixUtils.Multiply(inverse, new double[] { focus.X, focus.Y });
02775                 double[] centreVec = MatrixUtils.Multiply(inverse, new double[] { centre.X, centre.Y
    });
02776
02777                 focus = new Point(focusVec[0], focusVec[1]);
02778                 centre = new Point(centreVec[0], centreVec[1]);
02779             }
02780
02781             Avalonia.Media.RadialGradientBrush tbr = new Avalonia.Media.RadialGradientBrush()
02782             {
02783                 SpreadMethod = GradientSpreadMethod.Pad,
02784                 Center = new Avalonia.RelativePoint(centre.X, centre.Y,
    Avalonia.RelativeUnit.Absolute),
02785                 GradientOrigin = new Avalonia.RelativePoint(focus.X, focus.Y,
    Avalonia.RelativeUnit.Absolute),
02786                 Radius = brush.Radius / objectWidth
02787             };
02788
02789             Avalonia.Media.GradientStops stops = new Avalonia.Media.GradientStops();
02790             stops.AddRange(from el in brush.GradientStops select new
    Avalonia.Media.GradientStop(Color.FromArgb((byte)(el.Colour.A * 255), (byte)(el.Colour.R * 255),
    (byte)(el.Colour.G * 255), (byte)(el.Colour.B * 255)), el.Offset));
02791             tbr.GradientStops = stops;
02792
02793             return tbr;
02794         }
02795     }
02796 }
```

8.3 RenderingParameters.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019
00020 namespace VectSharp.Canvas
00021 {
00022     internal class RenderingParameters :  IEquatable<RenderingParameters>
00023     {
00024         public static float Tolerance = 1e-5f;
00025         public float Left { get; }
00026         public float Top { get; }
```

```
00027         public float Width { get; }
00028         public float Height { get; }
00029         public float Scale { get; }
00030         public int RenderWidth { get; }
00031         public int RenderHeight { get; }
00032 
00033         public RenderingParameters(float left, float top, float width, float height, float scale, int
      renderWidth, int renderHeight)
00034         {
00035             this.Left = left;
00036             this.Top = top;
00037             this.Width = width;
00038             this.Height = height;
00039             this.Scale = scale;
00040             this.RenderWidth = renderWidth;
00041             this.RenderHeight = renderHeight;
00042         }
00043 
00044         public RenderingParameters Clone()
00045         {
00046             return new RenderingParameters(this.Left, this.Top, this.Width, this.Height, this.Scale,
      this.RenderWidth, this.RenderHeight);
00047         }
00048 
00049         public bool Equals(RenderingParameters other)
00050         {
00051             if (!object.ReferenceEquals(other, null))
00052             {
00053                 return this.Scale == other.Scale && this.Left == other.Left && this.Top == other.Top
      && this.Width == other.Width && this.Height == other.Height && this.RenderWidth == other.RenderWidth
      && this.RenderHeight == other.RenderHeight;
00054             }
00055             else
00056             {
00057                 return false;
00058             }
00059         }
00060 
00061         public override bool Equals(object obj)
00062         {
00063             if (obj is RenderingParameters other)
00064             {
00065                 return this.Equals(other);
00066             }
00067             else
00068             {
00069                 return false;
00070             }
00071         }
00072 
00073         public override int GetHashCode()
00074         {
00075             int hash = 13;
00076 
00077             unchecked
00078             {
00079                 hash = (hash * 7) + this.Left.GetHashCode();
00080                 hash = (hash * 7) + this.Top.GetHashCode();
00081                 hash = (hash * 7) + this.Width.GetHashCode();
00082                 hash = (hash * 7) + this.Height.GetHashCode();
00083                 hash = (hash * 7) + this.Scale.GetHashCode();
00084                 hash = (hash * 7) + this.RenderWidth.GetHashCode();
00085                 hash = (hash * 7) + this.RenderHeight.GetHashCode();
00086             }
00087 
00088             return hash;
00089         }
00090 
00091         public static bool operator ==(RenderingParameters param1, RenderingParameters param2)
00092         {
00093             if (object.ReferenceEquals(param1, null))
00094             {
00095                 return object.ReferenceEquals(param2, null);
00096             }
00097             else
00098             {
00099                 return param1.Equals(param2);
00100             }
00101         }
00102 
00103         public static bool operator !=(RenderingParameters param1, RenderingParameters param2)
00104         {
00105             if (!object.ReferenceEquals(param1, null) && !object.ReferenceEquals(param2, null))
00106             {
00107                 return param1.Scale != param2.Scale || param1.Left != param2.Left || param1.Top !=
      param2.Top || param1.Width != param2.Width || param1.Height != param2.Height || param1.RenderWidth !=
      param2.RenderWidth || param1.RenderHeight != param2.RenderHeight;
```

```
00108             }
00109             else
00110             {
00111                 return !(object.ReferenceEquals(param1, null) && object.ReferenceEquals(param2,
      null));
00112             }
00113         }
00114 
00115         public bool GoodEnough(RenderingParameters other)
00116         {
00117             if (this.Scale == other.Scale)
00118             {
00119                 return (this.Left <= other.Left && (this.Left + this.Width - other.Left - other.Width)
      / (this.Left + this.Width + other.Left + other.Width) >= -Tolerance && this.Top <= other.Top &&
      (this.Top + this.Height - other.Top - other.Height) / (this.Top + this.Height + other.Top +
      other.Height) >= -Tolerance);
00120             }
00121             else
00122             {
00123                 return false;
00124             }
00125         }
00126     }
00127 }
```

8.4 SkiaBitmap.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using Avalonia;
00019 using Avalonia.Media;
00020 using Avalonia.Media.Imaging;
00021 using SkiaSharp;
00022 using System;
00023 using System.Linq.Expressions;
00024 using System.Reflection;
00025 
00026 namespace VectSharp.Canvas
00027 {
00028     internal class SkiaBitmap :  IImage, IDisposable
00029     {
00030         public SKCanvas SKCanvas { get; }
00031         public SKBitmap Bitmap { get; }
00032         public RenderTargetBitmap AvaloniaBitmap { get; set; }
00033         public Avalonia.Size Size { get; }
00034         public int Width { get; }
00035         public int Height { get; }
00036         public object Lock { get; }
00037 
00038 
00039         static Func<RenderTargetBitmap, SKBitmap> GetBitmap;
00040         static Func<RenderTargetBitmap, object> GetLock;
00041 
00042         static SkiaBitmap()
00043         {
00044             ParameterExpression bitmapParameter = Expression.Parameter(typeof(RenderTargetBitmap));
00045 
00046             PropertyInfo platformImplProperty = typeof(RenderTargetBitmap).GetProperty("PlatformImpl",
      BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.DeclaredOnly);
00047             MemberExpression getPlatformImpl = Expression.Property(bitmapParameter,
      platformImplProperty);
00048             MemberExpression getItem = Expression.Property(getPlatformImpl, "Item");
00049 
00050             Type renderTargetImpl =
      typeof(Avalonia.Skia.SkiaPlatform).Assembly.GetType("Avalonia.Skia.WriteableBitmapImpl");
00051 
00052             MemberExpression getBitmap = Expression.Field(Expression.Convert(getItem,
      renderTargetImpl), "_bitmap");
```

```
00053             MemberExpression getLock = Expression.Field(Expression.Convert(getItem, renderTargetImpl),
      "_lock");
00054
00055             GetBitmap = Expression.Lambda<Func<RenderTargetBitmap, SKBitmap»(getBitmap,
      bitmapParameter).Compile();
00056             GetLock = Expression.Lambda<Func<RenderTargetBitmap, object»(getLock,
      bitmapParameter).Compile();
00057         }
00058
00059         public SkiaBitmap(int width, int height)
00060         {
00061             this.AvaloniaBitmap = new RenderTargetBitmap(new PixelSize(width, height), new Vector(96,
      96));
00062             this.Lock = GetLock(this.AvaloniaBitmap);
00063             this.Bitmap = GetBitmap(this.AvaloniaBitmap);
00064
00065             this.SKCanvas = new SKCanvas(this.Bitmap);
00066             this.Size = new Avalonia.Size(width, height);
00067             this.Width = width;
00068             this.Height = height;
00069         }
00070
00071         private bool disposedValue;
00072
00073         public void Draw(DrawingContext context, Rect sourceRect, Rect destRect)
00074         {
00075             context.DrawImage(this.AvaloniaBitmap, sourceRect, destRect);
00076         }
00077
00078         protected virtual void Dispose(bool disposing)
00079         {
00080             if (!disposedValue)
00081             {
00082                 if (disposing)
00083                 {
00084                     this.SKCanvas?.Dispose();
00085                     this.AvaloniaBitmap?.Dispose();
00086                 }
00087                 disposedValue = true;
00088             }
00089         }
00090
00091         public void Dispose()
00092         {
00093             Dispose(disposing:  true);
00094             GC.SuppressFinalize(this);
00095         }
00096
00097         public void Indispose()
00098         {
00099             if (!disposedValue)
00100             {
00101                 this.SKCanvas?.Dispose();
00102                 this.disposedValue = true;
00103             }
00104         }
00105     }
00106 }
```

8.5 SKMultiLayerRenderCanvas.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using Avalonia;
00019 using Avalonia.Media;
00020 using Avalonia.Media.Imaging;
00021 using Avalonia.Skia;
00022 using SkiaSharp;
```

```
00023 using System;
00024 using System.Collections.Generic;
00025 using System.Linq;
00026 using System.Threading;
00027
00028 namespace VectSharp.Canvas
00029 {
00030     internal interface ISKRenderCanvas
00031     {
00032         void InvalidateDirty();
00033         void InvalidateZIndex();
00034     }
00035
00036 /// <summary>
00037 /// Represents a multi-threaded, triple-buffered canvas on which the image is drawn using SkiaSharp.
00038 /// </summary>
00039     public class SKMultiLayerRenderCanvas :  Avalonia.Controls.Control, IDisposable, ISKRenderCanvas
00040     {
00041 /// <summary>
00042 /// The width of the page that is rendered on this canvas.
00043 /// </summary>
00044         public double PageWidth { get; set; }
00045
00046 /// <summary>
00047 /// The height of the page that is rendered on this canvas.
00048 /// </summary>
00049         public double PageHeight { get; set; }
00050         private SKColor BackgroundColour;
00051
00052 /// <summary>
00053 /// Defines an overdraw margin for the clipping area.  Set this to a larger value if quickly moving the
00054 /// canvas around (e.g.  in a ZoomBorder) causes the edge of the image to disappear for a few milliseconds
00055 /// and that annoys you.  If <see cref="RelativePoint.Unit"/> is set to <see cref="RelativeUnit.Absolute"/>,
00056 /// the value corresponds to units on the <see cref="VectSharp.Page"/>; if it is set to <see cref="RelativeUnit.Relative"/>,
00057 /// it corresponds to a proportion of the visible area.
00058 /// </summary>
00059         public RelativePoint ClipMargin { get; set; } = new RelativePoint(0, 0, RelativeUnit.Absolute);
00060
00061 /// <summary>
00062 /// The list of render actions.  Each element in this list is itself a list, containing the actions that correspond to a layer in the image.
00063 /// </summary>
00064         public List<List<SKRenderAction>> RenderActions;
00065
00066 /// <summary>
00067 /// The list of transforms associated with each layer.
00068 /// </summary>
00069         public List<SKRenderAction> LayerTransforms;
00070
00071         private Dictionary<string, (SKBitmap, bool)> Images;
00072         private List<List<SKRenderAction>> TaggedRenderActions;
00073
00074 /// <summary>
00075 /// Create a new SKMultiLayerRenderCanvas from a <see cref="Document"/>, where each page represents a layer.
00076 /// </summary>
00077 /// <param name="document">The document containing the layers as <see cref="Page"/>s.</param>
00078 /// <param name="layerTransforms">A list of transforms associated with each layer.  This list should contain the same number of elements as the number of pages in <paramref name="document"/>.  This is useful to manipulate the position of each layer individually.  If this is null, an identity transform is applied to each layer.</param>
00079 /// <param name="backgroundColour">The background colour of the canvas.</param>
00080 /// <param name="width">The width of the canvas and the pages it contains.</param>
00081 /// <param name="height">The height of the canvas and the pages it contains.</param>
00082         public SKMultiLayerRenderCanvas(Document document, Colour backgroundColour, double width, double height, List<SKRenderAction> layerTransforms = null) :  this(document.Pages, backgroundColour, width, height, layerTransforms) { }
00083
00084 /// <summary>
00085 /// Create a new SKMultiLayerRenderCanvas from a collection of <see cref="Page"/>s, each representing a layer.
00086 /// </summary>
00087 /// <param name="layers">The contents of the canvas.  Each element in this list represents a layer.</param>
00088 /// <param name="layerTransforms">A list of transforms associated with each layer.  This list should contain the same number of elements as <paramref name="layers"/>.  This is useful to manipulate the position of each layer individually.  If this is null, an identity transform is applied to each layer.</param>
00089 /// <param name="backgroundColour">The background colour of the canvas.</param>
00090 /// <param name="width">The width of the canvas and the pages it contains.</param>
00091 /// <param name="height">The height of the canvas and the pages it contains.</param>
00092         public SKMultiLayerRenderCanvas(IEnumerable<Page> layers, Colour backgroundColour, double
```

```
      width, double height, List<SKRenderAction> layerTransforms = null)
00093         {
00094             List<SKRenderContext> contents = (from el in layers select
      el.CopyToSKRenderContext()).ToList();
00095             List<SKRenderAction> contentTransforms;
00096
00097             if (layerTransforms == null)
00098             {
00099                 contentTransforms = (from el in Enumerable.Range(0, contents.Count) select
      SKRenderAction.TransformAction(SKMatrix.Identity)).ToList();
00100             }
00101             else
00102             {
00103                 contentTransforms = layerTransforms;
00104             }
00105
00106             UpdateWith(contents, contentTransforms, backgroundColour, width, height);
00107
00108             this.Width = width;
00109             this.Height = height;
00110
00111             this.PointerPressed += this.PointerPressedAction;
00112             this.PointerReleased += this.PointerReleasedAction;
00113             this.PointerMoved += this.PointerMoveAction;
00114             this.PointerExited += this.PointerLeaveAction;
00115         }
00116
00117 /// <summary>
00118 /// Create a new SKMultiLayerRenderCanvas from a list of SKRenderContexts, each representing a layer.
00119 /// </summary>
00120 /// <param name="contents">The contents of the canvas.  Each element in this list represents a layer.
      A Page can be converded to a SKRenderContext through the CopyToSKRenderContext method.</param>
00121 /// <param name="contentTransforms">A list of transforms associated with each layer.  This list should
      contain the same number of elements as <paramref name="contents"/>.  This is useful to manipulate the
      position of each layer individually.</param>
00122 /// <param name="backgroundColour">The background colour of the canvas.</param>
00123 /// <param name="width">The width of the canvas and the page it contains.</param>
00124 /// <param name="height">The height of the canvas and the page it contains.</param>
00125         public SKMultiLayerRenderCanvas(List<SKRenderContext> contents, List<SKRenderAction>
      contentTransforms, Colour backgroundColour, double width, double height)
00126         {
00127             UpdateWith(contents, contentTransforms, backgroundColour, width, height);
00128
00129             this.Width = width;
00130             this.Height = height;
00131
00132             this.PointerPressed += this.PointerPressedAction;
00133             this.PointerReleased += this.PointerReleasedAction;
00134             this.PointerMoved += this.PointerMoveAction;
00135             this.PointerExited += this.PointerLeaveAction;
00136         }
00137
00138 /// <summary>
00139 /// Replace the contents of the SKMultiLayerRenderCanvas with the specified layers.
00140 /// </summary>
00141 /// <param name="contents">The contents of the canvas.  Each element in this list represents a layer.
      A Page can be converded to a SKRenderContext through the CopyToSKRenderContext method.</param>
00142 /// <param name="contentTransforms">A list of transforms associated with each layer.  This list should
      contain the same number of elements as <paramref name="contents"/>.  This is useful to manipulate the
      position of each layer individually.</param>
00143 /// <param name="backgroundColour">The background colour of the canvas.</param>
00144 /// <param name="width">The width of the canvas and the page it contains.</param>
00145 /// <param name="height">The height of the canvas and the page it contains.</param>
00146         public void UpdateWith(List<SKRenderContext> contents, List<SKRenderAction> contentTransforms,
      Colour backgroundColour, double width, double height)
00147         {
00148             lock (RenderLock)
00149             {
00150                 if (this.RenderActions == null)
00151                 {
00152                     this.RenderActions = new List<List<SKRenderAction»();
00153                 }
00154                 else
00155                 {
00156                     this.RenderActions.Clear();
00157                 }
00158
00159                 if (this.LayerTransforms == null)
00160                 {
00161                     this.LayerTransforms = new List<SKRenderAction>();
00162                 }
00163                 else
00164                 {
00165                     this.LayerTransforms.Clear();
00166                 }
00167
00168                 this.PageWidth = width;
```

```
00169                 this.PageHeight = height;
00170                 this.BackgroundColour = backgroundColour.ToSKColor();
00171 
00172                 if (this.Images == null)
00173                 {
00174                     Images = new Dictionary<string, (SKBitmap, bool)>();
00175                 }
00176                 else
00177                 {
00178                     Images.Clear();
00179                 }
00180 
00181                 for (int i = 0; i < contents.Count; i++)
00182                 {
00183                     this.LayerTransforms.Add(contentTransforms[i]);
00184                     this.RenderActions.Add(contents[i].SKRenderActions);
00185 
00186                     foreach (KeyValuePair<string, (SKBitmap, bool)> kvp in contents[i].Images)
00187                     {
00188                         this.Images[kvp.Key] = kvp.Value;
00189                     }
00190                 }
00191 
00192                 if (this.TaggedRenderActions == null)
00193                 {
00194                     this.TaggedRenderActions = new List<List<SKRenderAction»();
00195                 }
00196                 else
00197                 {
00198                     this.TaggedRenderActions.Clear();
00199                 }
00200 
00201 
00202                 for (int i = this.RenderActions.Count - 1; i >= 0; i--)
00203                 {
00204                     TaggedRenderActions.Add(new List<SKRenderAction>());
00205 
00206                     for (int j = this.RenderActions[i].Count - 1; j >= 0; j--)
00207                     {
00208                         RenderActions[i][j].InternalParent = this;
00209                         if (!string.IsNullOrEmpty(this.RenderActions[i][j].Tag))
00210                         {
00211                             TaggedRenderActions[this.RenderActions.Count - 1 -
    i].Add(this.RenderActions[i][j]);
00212                         }
00213                     }
00214                 }
00215             }
00216         }
00217 
00218 /// <summary>
00219 /// Replace a single layer with the specified content.
00220 /// </summary>
00221 /// <param name="layer">The index of the layer to replace.</param>
00222 /// <param name="newContent">The new contents of the layer.  A Page can be converded to a
    SKRenderContext through the CopyToSKRenderContext method.</param>
00223 /// <param name="newTransform">The new transform for the layer.</param>
00224         public void UpdateLayer(int layer, SKRenderContext newContent, SKRenderAction newTransform)
00225         {
00226             lock (RenderLock)
00227             {
00228                 this.LayerTransforms[layer] = newTransform;
00229                 this.RenderActions[layer] = newContent.SKRenderActions;
00230 
00231                 foreach (KeyValuePair<string, (SKBitmap, bool)> kvp in newContent.Images)
00232                 {
00233                     this.Images[kvp.Key] = kvp.Value;
00234                 }
00235 
00236                 TaggedRenderActions[this.RenderActions.Count - 1 - layer].Clear();
00237 
00238                 for (int j = this.RenderActions[layer].Count - 1; j >= 0; j--)
00239                 {
00240                     RenderActions[layer][j].InternalParent = this;
00241                     if (!string.IsNullOrEmpty(this.RenderActions[layer][j].Tag))
00242                     {
00243                         TaggedRenderActions[this.RenderActions.Count - 1 -
    layer].Add(this.RenderActions[layer][j]);
00244                     }
00245                 }
00246             }
00247 
00248             this.InvalidateDirty();
00249         }
00250 
00251 /// <summary>
00252 /// Add a new layer to the image.
```

```
00253 /// </summary>
00254 /// <param name="newContent">The contents of the new layer.  A Page can be converded to a
      SKRenderContext through the CopyToSKRenderContext method.</param>
00255 /// <param name="newTransform">The transform for the new layer.</param>
00256         public void AddLayer(SKRenderContext newContent, SKRenderAction newTransform)
00257         {
00258             lock (RenderLock)
00259             {
00260                 this.LayerTransforms.Add(newTransform);
00261                 this.RenderActions.Add(newContent.SKRenderActions);
00262 
00263                 foreach (KeyValuePair<string, (SKBitmap, bool)> kvp in newContent.Images)
00264                 {
00265                     this.Images[kvp.Key] = kvp.Value;
00266                 }
00267 
00268                 TaggedRenderActions.Insert(0, new List<SKRenderAction>());
00269 
00270                 for (int j = this.RenderActions[this.RenderActions.Count - 1].Count - 1; j >= 0; j--)
00271                 {
00272                     RenderActions[this.RenderActions.Count - 1][j].InternalParent = this;
00273                     if (!string.IsNullOrEmpty(this.RenderActions[this.RenderActions.Count -
      1][j].Tag))
00274                     {
00275                         TaggedRenderActions[0].Add(this.RenderActions[this.RenderActions.Count -
      1][j]);
00276                     }
00277                 }
00278             }
00279 
00280             this.InvalidateDirty();
00281         }
00282 
00283 /// <summary>
00284 /// Insert a new layer at the specified index.
00285 /// </summary>
00286 /// <param name="index">The position at which the new layer will be inserted.</param>
00287 /// <param name="newContent">The contents of the new layer.</param>
00288 /// <param name="newTransform">The transform for the new layer.</param>
00289         public void InsertLayer(int index, SKRenderContext newContent, SKRenderAction newTransform)
00290         {
00291             lock (RenderLock)
00292             {
00293                 this.LayerTransforms.Insert(index, newTransform);
00294                 this.RenderActions.Insert(index, newContent.SKRenderActions);
00295 
00296                 foreach (KeyValuePair<string, (SKBitmap, bool)> kvp in newContent.Images)
00297                 {
00298                     this.Images[kvp.Key] = kvp.Value;
00299                 }
00300 
00301                 TaggedRenderActions.Insert(this.RenderActions.Count - 1 - index, new
      List<SKRenderAction>());
00302 
00303                 for (int j = this.RenderActions[index].Count - 1; j >= 0; j--)
00304                 {
00305                     RenderActions[index][j].InternalParent = this;
00306                     if (!string.IsNullOrEmpty(this.RenderActions[index][j].Tag))
00307                     {
00308                         TaggedRenderActions[this.RenderActions.Count - 1 -
      index].Add(this.RenderActions[index][j]);
00309                     }
00310                 }
00311             }
00312 
00313             this.InvalidateDirty();
00314         }
00315 
00316 /// <summary>
00317 /// Remove the specified layer from the image.
00318 /// </summary>
00319 /// <param name="layer">The index of the layer to remove.</param>
00320         public void RemoveLayer(int layer)
00321         {
00322             lock (RenderLock)
00323             {
00324                 this.LayerTransforms[layer].Dispose();
00325                 this.LayerTransforms.RemoveAt(layer);
00326 
00327                 for (int i = 0; i < this.RenderActions[layer].Count; i++)
00328                 {
00329                     this.RenderActions[layer][i].Dispose();
00330                 }
00331                 this.RenderActions.RemoveAt(layer);
00332 
00333                 TaggedRenderActions.RemoveAt(this.TaggedRenderActions.Count - 1 - layer);
00334             }
```

```
00335
00336             this.InvalidateDirty();
00337         }
00338
00339 /// <summary>
00340 /// Switch the position of the two specified layers.
00341 /// </summary>
00342 /// <param name="layer1">The index of the first layer to switch.</param>
00343 /// <param name="layer2">The index of the second layer to switch.</param>
00344         public void SwitchLayers(int layer1, int layer2)
00345         {
00346             lock (RenderLock)
00347             {
00348                 var temp = this.LayerTransforms[layer1];
00349                 this.LayerTransforms[layer1] = this.LayerTransforms[layer2];
00350                 this.LayerTransforms[layer2] = temp;
00351
00352                 var temp2 = this.RenderActions[layer1];
00353                 this.RenderActions[layer1] = this.RenderActions[layer2];
00354                 this.RenderActions[layer2] = temp2;
00355
00356                 var temp3 = this.TaggedRenderActions[this.RenderActions.Count - 1 - layer1];
00357                 this.TaggedRenderActions[this.RenderActions.Count - 1 - layer1] =
    this.TaggedRenderActions[this.RenderActions.Count - 1 - layer2];
00358                 this.TaggedRenderActions[this.RenderActions.Count - 1 - layer2] = temp3;
00359             }
00360
00361             this.InvalidateDirty();
00362         }
00363
00364 /// <summary>
00365 /// Move the specified layer to the specified position, shifting all other layers as necessary.
00366 /// </summary>
00367 /// <param name="oldIndex">The current index of the layer to move.</param>
00368 /// <param name="newIndex">The final index of the layer.  Layers after this will be shifted by 1 in
    order to accommodate the moved layer.</param>
00369         public void MoveLayer(int oldIndex, int newIndex)
00370         {
00371             lock (RenderLock)
00372             {
00373                 var temp = this.LayerTransforms[oldIndex];
00374                 this.LayerTransforms.RemoveAt(oldIndex);
00375                 this.LayerTransforms.Insert(newIndex, temp);
00376
00377                 var temp2 = this.RenderActions[oldIndex];
00378                 this.RenderActions.RemoveAt(oldIndex);
00379                 this.RenderActions.Insert(newIndex, temp2);
00380
00381                 var temp3 = this.TaggedRenderActions[this.RenderActions.Count - 1 - oldIndex];
00382                 this.TaggedRenderActions.RemoveAt(this.RenderActions.Count - 1 - oldIndex);
00383                 this.TaggedRenderActions.Insert(this.RenderActions.Count - 1 - newIndex, temp3);
00384             }
00385
00386             this.InvalidateDirty();
00387         }
00388
00389
00390         private SKRenderAction CurrentPressedAction = null;
00391         private void PointerPressedAction(object sender, Avalonia.Input.PointerPressedEventArgs e)
00392         {
00393             SKPoint position = e.GetPosition(this).ToSKPoint();
00394
00395             for (int i = 0; i < TaggedRenderActions.Count; i++)
00396             {
00397                 bool found = false;
00398
00399                 for (int j = 0; j < TaggedRenderActions[i].Count; j++)
00400                 {
00401                     if (TaggedRenderActions[i][j].LastRenderedGlobalHitTestPath?.Contains(position.X,
    position.Y) == true)
00402                     {
00403
00404                         if (TaggedRenderActions[i][j].ActionType == SKRenderAction.ActionTypes.Path)
00405                         {
00406                             CurrentPressedAction = TaggedRenderActions[i][j];
00407                             TaggedRenderActions[i][j].FirePointerPressed(e);
00408                             found = true;
00409                             break;
00410                         }
00411                         else if (TaggedRenderActions[i][j].ActionType ==
    SKRenderAction.ActionTypes.Text)
00412                         {
00413                             CurrentPressedAction = TaggedRenderActions[i][j];
00414                             TaggedRenderActions[i][j].FirePointerPressed(e);
00415                             found = true;
00416                             break;
00417                         }
```

```
00418                         else if (TaggedRenderActions[i][j].ActionType ==
      SKRenderAction.ActionTypes.RasterImage)
00419                         {
00420                             CurrentPressedAction = TaggedRenderActions[i][j];
00421                             TaggedRenderActions[i][j].FirePointerPressed(e);
00422                             found = true;
00423                             break;
00424                         }
00425                     }
00426                 }
00427 
00428                 if (found)
00429                 {
00430                     break;
00431                 }
00432             }
00433         }
00434 
00435         private void PointerReleasedAction(object sender, Avalonia.Input.PointerReleasedEventArgs e)
00436         {
00437             if (CurrentPressedAction != null)
00438             {
00439                 if (!CurrentPressedAction.Disposed)
00440                 {
00441                     CurrentPressedAction.FirePointerReleased(e);
00442                 }
00443                 CurrentPressedAction = null;
00444             }
00445             else
00446             {
00447                 SKPoint position = e.GetPosition(this).ToSKPoint();
00448 
00449                 for (int i = 0; i < TaggedRenderActions.Count; i++)
00450                 {
00451                     bool found = false;
00452 
00453                     for (int j = 0; j < TaggedRenderActions[i].Count; j++)
00454                     {
00455                         if
      (TaggedRenderActions[i][j].LastRenderedGlobalHitTestPath?.Contains(position.X, position.Y) == true)
00456                         {
00457                             if (TaggedRenderActions[i][j].ActionType ==
      SKRenderAction.ActionTypes.Path)
00458                             {
00459                                 TaggedRenderActions[i][j].FirePointerReleased(e);
00460                                 found = true;
00461                                 break;
00462                             }
00463                             else if (TaggedRenderActions[i][j].ActionType ==
      SKRenderAction.ActionTypes.Text)
00464                             {
00465                                 TaggedRenderActions[i][j].FirePointerReleased(e);
00466                                 found = true;
00467                                 break;
00468                             }
00469                             else if (TaggedRenderActions[i][j].ActionType ==
      SKRenderAction.ActionTypes.RasterImage)
00470                             {
00471                                 TaggedRenderActions[i][j].FirePointerReleased(e);
00472                                 found = true;
00473                                 break;
00474                             }
00475                         }
00476                     }
00477 
00478                     if (found)
00479                     {
00480                         break;
00481                     }
00482                 }
00483             }
00484         }
00485 
00486         private SKRenderAction CurrentOverAction = null;
00487         private void PointerMoveAction(object sender, Avalonia.Input.PointerEventArgs e)
00488         {
00489             SKPoint position = e.GetPosition(this).ToSKPoint();
00490 
00491             bool found = false;
00492 
00493             for (int i = 0; i < TaggedRenderActions.Count; i++)
00494             {
00495                 for (int j = 0; j < TaggedRenderActions[i].Count; j++)
00496                 {
00497                     if (TaggedRenderActions[i][j].LastRenderedGlobalHitTestPath?.Contains(position.X,
      position.Y) == true)
00498                     {
```

```
00499                         if (TaggedRenderActions[i][j].ActionType == SKRenderAction.ActionTypes.Path)
00500                         {
00501                             found = true;
00502 
00503                             if (CurrentOverAction != TaggedRenderActions[i][j])
00504                             {
00505                                 if (CurrentOverAction != null && !CurrentOverAction.Disposed)
00506                                 {
00507                                     CurrentOverAction.FirePointerExited(e);
00508                                 }
00509                                 CurrentOverAction = TaggedRenderActions[i][j];
00510                                 CurrentOverAction.FirePointerEntered(e);
00511                             }
00512 
00513                             break;
00514                         }
00515                         else if (TaggedRenderActions[i][j].ActionType ==
      SKRenderAction.ActionTypes.Text)
00516                         {
00517                             found = true;
00518 
00519                             if (CurrentOverAction != TaggedRenderActions[i][j])
00520                             {
00521                                 if (CurrentOverAction != null && !CurrentOverAction.Disposed)
00522                                 {
00523                                     CurrentOverAction.FirePointerExited(e);
00524                                 }
00525                                 CurrentOverAction = TaggedRenderActions[i][j];
00526                                 CurrentOverAction.FirePointerEntered(e);
00527                             }
00528 
00529                             break;
00530                         }
00531                         else if (TaggedRenderActions[i][j].ActionType ==
      SKRenderAction.ActionTypes.RasterImage)
00532                         {
00533                             found = true;
00534 
00535                             if (CurrentOverAction != TaggedRenderActions[i][j])
00536                             {
00537                                 if (CurrentOverAction != null && !CurrentOverAction.Disposed)
00538                                 {
00539                                     CurrentOverAction.FirePointerExited(e);
00540                                 }
00541                                 CurrentOverAction = TaggedRenderActions[i][j];
00542                                 CurrentOverAction.FirePointerEntered(e);
00543                             }
00544 
00545                             break;
00546                         }
00547                     }
00548                 }
00549 
00550                 if (found)
00551                 {
00552                     break;
00553                 }
00554             }
00555 
00556             if (!found)
00557             {
00558                 if (CurrentOverAction != null && !CurrentOverAction.Disposed)
00559                 {
00560                     CurrentOverAction.FirePointerExited(e);
00561                 }
00562                 CurrentOverAction = null;
00563             }
00564         }
00565 
00566         private void PointerLeaveAction(object sender, Avalonia.Input.PointerEventArgs e)
00567         {
00568             if (CurrentOverAction != null && !CurrentOverAction.Disposed)
00569             {
00570                 CurrentOverAction.FirePointerExited(e);
00571             }
00572             CurrentOverAction = null;
00573         }
00574 
00575 /// <inheritdoc/>
00576         protected override void OnAttachedToVisualTree(VisualTreeAttachmentEventArgs e)
00577         {
00578             base.OnAttachedToVisualTree(e);
00579 
00580             StartRenderingThread();
00581         }
00582 
00583 /// <inheritdoc/>
```

```
00584         protected override void OnDetachedFromVisualTree(VisualTreeAttachmentEventArgs e)
00585         {
00586             StopRenderingThread();
00587         }
00588 
00589         private void StopRenderingThread()
00590         {
00591             DisposedHandle.Set();
00592         }
00593 
00594         private EventWaitHandle DisposedHandle;
00595         private EventWaitHandle RenderRequestedHandle;
00596 
00597         private RenderingParameters RenderingRequest = null;
00598         private readonly object RenderingRequestLock = new object();
00599 
00600         private void StartRenderingThread()
00601         {
00602             DisposedHandle = new EventWaitHandle(false, EventResetMode.ManualReset);
00603             RenderRequestedHandle = new EventWaitHandle(false, EventResetMode.ManualReset);
00604 
00605             Thread renderingThread = new Thread(async () =>
00606             {
00607                 bool finished = false;
00608                 WaitHandle[] handles = new WaitHandle[] { DisposedHandle, RenderRequestedHandle };
00609 
00610                 while (!finished)
00611                 {
00612                     int handle = EventWaitHandle.WaitAny(handles);
00613 
00614                     if (handle == 0)
00615                     {
00616                         finished = true;
00617                     }
00618                     else if (handle == 1)
00619                     {
00620                         RenderingParameters requestParams;
00621 
00622                         lock (RenderingRequestLock)
00623                         {
00624                             requestParams = RenderingRequest.Clone();
00625                             RenderRequestedHandle.Reset();
00626                         }
00627 
00628                         SKCanvas canvas;
00629 
00630                         if (BackBuffer == null || BackBufferRenderingParams == null ||
      requestParams.RenderWidth > BackBufferRenderingParams.RenderWidth || requestParams.RenderHeight >
      BackBufferRenderingParams.RenderHeight)
00631                         {
00632                             await Avalonia.Threading.Dispatcher.UIThread.InvokeAsync(() =>
00633                             {
00634                                 SkiaBitmap tempBufferReference = BackBuffer;
00635 
00636                                 _ = Avalonia.Threading.Dispatcher.UIThread.InvokeAsync(() =>
00637                                 {
00638                                     tempBufferReference?.Dispose();
00639                                 }, Avalonia.Threading.DispatcherPriority.Send);
00640 
00641                                 BackBuffer = new SkiaBitmap(requestParams.RenderWidth,
      requestParams.RenderHeight);
00642                                 BackBufferSkiaCanvas = BackBuffer.SKCanvas;
00643                             }, Avalonia.Threading.DispatcherPriority.MaxValue);
00644                         }
00645 
00646                         canvas = BackBufferSkiaCanvas;
00647 
00648                         canvas.Save();
00649 
00650                         canvas.Scale(requestParams.Scale);
00651                         canvas.Translate(-requestParams.Left, -requestParams.Top);
00652 
00653                         canvas.Clear(BackgroundColour);
00654 
00655                         canvas.ClipRect(new SKRect(requestParams.Left, requestParams.Top,
      requestParams.Left + requestParams.Width, requestParams.Top + requestParams.Height));
00656 
00657                         RenderImage(canvas);
00658 
00659                         canvas.RestoreToCount(-1);
00660 
00661                         BackBufferRenderingParams = requestParams;
00662 
00663                         lock (FrontBufferLock)
00664                         {
00665                             SkiaBitmap tempFrontReference = FrontBuffer;
00666                             RenderingParameters tempFrontRenderingParameters =
```

```
    FrontBufferRenderingParams;
00667                               SKCanvas tempFrontCanvas = FrontBufferSkiaCanvas;
00668
00669                               FrontBuffer = BackBuffer;
00670                               FrontBufferRenderingParams = BackBufferRenderingParams;
00671                               FrontBufferSkiaCanvas = BackBufferSkiaCanvas;
00672
00673                               BackBuffer = BackBuffer2;
00674                               BackBufferRenderingParams = BackBufferRenderingParams2;
00675                               BackBufferSkiaCanvas = BackBufferSkiaCanvas2;
00676
00677                               BackBuffer2 = tempFrontReference;
00678                               BackBufferRenderingParams2 = tempFrontRenderingParameters;
00679                               BackBufferSkiaCanvas2 = tempFrontCanvas;
00680                           }
00681
00682                           await Avalonia.Threading.Dispatcher.UIThread.InvokeAsync(() =>
00683                           {
00684                               base.InvalidateVisual();
00685                           }, Avalonia.Threading.DispatcherPriority.MaxValue);
00686                       }
00687                   }
00688               });
00689
00690               renderingThread.Start();
00691           }
00692
00693 /// <summary>
00694 /// A lock for the rendering loop.  The public methods of this class already lock on this, but you may
    need it if you want to directly manipulate the contents of the canvas.
00695 /// </summary>
00696           public object RenderLock = new object();
00697
00698 /// <summary>
00699 /// Render the image to a bitmap at the specified resolution.
00700 /// </summary>
00701 /// <param name="width">The width of the rendered image.  Note that the actual width of the returned
    image might be lower than this, depending on the aspect ratio of the image.</param>
00702 /// <param name="height">The height of the rendered image.  Note that the actual height of the
    returned image might be lower than this, depending on the aspect ratio of the image.</param>
00703 /// <param name="background">The background colour for the image.  If this is <see langword="null" />,
    the current background colour is used.</param>
00704 /// <returns>A <see cref="RenderTargetBitmap"/> containing the image rendered at the specified
    resolution.</returns>
00705           public RenderTargetBitmap RenderAtResolution(int width, int height, SKColor?  background =
    null)
00706           {
00707               SKColor realBackground = background ??  this.BackgroundColour;
00708
00709               double scale = Math.Min(width / this.PageWidth, height / this.PageHeight);
00710
00711               width = (int)Math.Round(this.PageWidth * scale);
00712               height = (int)Math.Round(this.PageHeight * scale);
00713
00714               SkiaBitmap skBmp = new SkiaBitmap(width, height);
00715               SKCanvas canvas = skBmp.SKCanvas;
00716
00717               canvas.Clear(realBackground);
00718
00719               canvas.Save();
00720
00721               canvas.Scale((float)scale);
00722
00723               RenderImage(canvas);
00724
00725               canvas.RestoreToCount(-1);
00726
00727               skBmp.Indispose();
00728
00729               return skBmp.AvaloniaBitmap;
00730           }
00731
00732           private void RenderImage(SKCanvas canvas)
00733           {
00734               lock (RenderLock)
00735               {
00736                   for (int i = 0; i < this.RenderActions.Count; i++)
00737                   {
00738                       canvas.Save();
00739
00740                       if (this.LayerTransforms[i].ActionType == SKRenderAction.ActionTypes.Transform)
00741                       {
00742                           SKMatrix mat = this.LayerTransforms[i].Transform.Value;
00743                           canvas.Concat(ref mat);
00744                       }
00745
00746                       RenderLayer(canvas, i);
```

```
00747                     canvas.Restore();
00748                 }
00749             }
00750         }
00751 
00752         private void RenderLayer(SKCanvas canvas, int layer)
00753         {
00754             bool updateHitTests = this.TaggedRenderActions[this.TaggedRenderActions.Count - 1 -
      layer].Count > 0;
00755 
00756             HashSet<uint> ZIndices = new HashSet<uint>();
00757 
00758             canvas.Save();
00759 
00760             SKMatrix invertedInitialTransform;
00761 
00762             if (this.LayerTransforms[layer].ActionType == SKRenderAction.ActionTypes.Transform)
00763             {
00764                 invertedInitialTransform =
      (canvas.TotalMatrix.PreConcat(this.LayerTransforms[layer].Transform.Value.Invert())).Invert();
00765             }
00766             else
00767             {
00768                 invertedInitialTransform = canvas.TotalMatrix.Invert();
00769             }
00770 
00771             SKPath clipPath = null;
00772             Stack<SKPath> clipPaths = new Stack<SKPath>();
00773             clipPaths.Push(null);
00774 
00775             for (int i = 0; i < this.RenderActions[layer].Count; i++)
00776             {
00777                 ZIndices.Add(this.RenderActions[layer][i].ZIndex);
00778 
00779                 if (this.RenderActions[layer][i].ActionType == SKRenderAction.ActionTypes.Clip)
00780                 {
00781                     canvas.ClipPath(this.RenderActions[layer][i].Path, antialias:  true);
00782 
00783                     if (updateHitTests)
00784                     {
00785                         if (clipPath == null)
00786                         {
00787                             clipPath = this.RenderActions[layer][i].Path.Clone();
00788 
      clipPath.Transform(canvas.TotalMatrix.PostConcat(invertedInitialTransform));
00789                         }
00790                         else
00791                         {
00792                             using (SKPath tempPath = this.RenderActions[layer][i].Path.Clone())
00793                             {
00794 
      tempPath.Transform(canvas.TotalMatrix.PostConcat(invertedInitialTransform));
00795                                 clipPath.Op(tempPath, SKPathOp.Intersect);
00796                             }
00797                         }
00798                     }
00799                 }
00800                 else if (this.RenderActions[layer][i].ActionType ==
      SKRenderAction.ActionTypes.Restore)
00801                 {
00802                     canvas.Restore();
00803 
00804                     if (updateHitTests)
00805                     {
00806                         clipPath = clipPaths.Pop();
00807                     }
00808                 }
00809                 else if (this.RenderActions[layer][i].ActionType == SKRenderAction.ActionTypes.Save)
00810                 {
00811                     canvas.Save();
00812 
00813                     if (updateHitTests)
00814                     {
00815                         clipPaths.Push(clipPath?.Clone());
00816                     }
00817                 }
00818                 else if (this.RenderActions[layer][i].ActionType ==
      SKRenderAction.ActionTypes.Transform)
00819                 {
00820                     SKMatrix mat = this.RenderActions[layer][i].Transform.Value;
00821                     canvas.Concat(ref mat);
00822                 }
00823                 else
00824                 {
00825                     if (this.RenderActions[layer][i].ZIndex == 0)
00826                     {
00827 
```

```
00828                         if (this.RenderActions[layer][i].ActionType == SKRenderAction.ActionTypes.Path
      && this.RenderActions[layer][i].Path != null)
00829                         {
00830                             canvas.DrawPath(this.RenderActions[layer][i].Path,
      this.RenderActions[layer][i].Paint);
00831                         }
00832                         else if (this.RenderActions[layer][i].ActionType ==
      SKRenderAction.ActionTypes.Text)
00833                         {
00834                             canvas.DrawText(this.RenderActions[layer][i].Text,
      this.RenderActions[layer][i].TextX, this.RenderActions[layer][i].TextY,
      this.RenderActions[layer][i].Font, this.RenderActions[layer][i].Paint);
00835                         }
00836                         else if (this.RenderActions[layer][i].ActionType ==
      SKRenderAction.ActionTypes.RasterImage)
00837                         {
00838                             (SKBitmap image, bool interpolate) =
      this.Images[this.RenderActions[layer][i].ImageId];
00839 
00840                             SKPaint paint;
00841 
00842                             if (!interpolate)
00843                             {
00844                                 paint = null;
00845                             }
00846                             else
00847                             {
00848                                 paint = new SKPaint() { FilterQuality = SKFilterQuality.Medium };
00849                             }
00850 
00851                             canvas.DrawBitmap(image, this.RenderActions[layer][i].ImageSource.Value,
      this.RenderActions[layer][i].ImageDestination.Value, paint);
00852 
00853                             paint?.Dispose();
00854                         }
00855                     }
00856 
00857                     if (updateHitTests && this.RenderActions[layer][i].Tag != null &&
      this.RenderActions[layer][i].HitTestPath != null)
00858                     {
00859                         SKPath hitTestPath = this.RenderActions[layer][i].HitTestPath.Clone();
00860 
      hitTestPath.Transform(canvas.TotalMatrix.PostConcat(invertedInitialTransform));
00861 
00862                         if (clipPath != null)
00863                         {
00864                             hitTestPath.Op(clipPath, SKPathOp.Intersect);
00865                         }
00866 
00867                         this.RenderActions[layer][i].LastRenderedGlobalHitTestPath = hitTestPath;
00868                     }
00869                 }
00870             }
00871 
00872             canvas.Restore();
00873 
00874             uint[] sortedIndices = ZIndices.OrderBy(x => x).ToArray();
00875 
00876             if (sortedIndices.Length > 1 || sortedIndices[0] != 0)
00877             {
00878                 for (int j = 0; j < sortedIndices.Length; j++)
00879                 {
00880                     canvas.Save();
00881 
00882                     for (int i = 0; i < this.RenderActions[layer].Count; i++)
00883                     {
00884                         if (this.RenderActions[layer][i].ActionType ==
      SKRenderAction.ActionTypes.Clip)
00885                         {
00886                             canvas.ClipPath(this.RenderActions[layer][i].Path, antialias:  true);
00887                         }
00888                         else if (this.RenderActions[layer][i].ActionType ==
      SKRenderAction.ActionTypes.Restore)
00889                         {
00890                             canvas.Restore();
00891                         }
00892                         else if (this.RenderActions[layer][i].ActionType ==
      SKRenderAction.ActionTypes.Save)
00893                         {
00894                             canvas.Save();
00895                         }
00896                         else if (this.RenderActions[layer][i].ActionType ==
      SKRenderAction.ActionTypes.Transform)
00897                         {
00898                             SKMatrix mat = this.RenderActions[layer][i].Transform.Value;
00899                             canvas.Concat(ref mat);
00900                         }
```

```
00901                         else if (sortedIndices[j] != 0 && this.RenderActions[layer][i].ZIndex ==
      sortedIndices[j])
00902                         {
00903                             if (this.RenderActions[layer][i].ActionType ==
      SKRenderAction.ActionTypes.Path && this.RenderActions[layer][i].Path != null)
00904                             {
00905                                 canvas.DrawPath(this.RenderActions[layer][i].Path,
      this.RenderActions[layer][i].Paint);
00906                             }
00907                             else if (this.RenderActions[layer][i].ActionType ==
      SKRenderAction.ActionTypes.Text)
00908                             {
00909                                 canvas.DrawText(this.RenderActions[layer][i].Text,
      this.RenderActions[layer][i].TextX, this.RenderActions[layer][i].TextY,
      this.RenderActions[layer][i].Font, this.RenderActions[layer][i].Paint);
00910                             }
00911                             else if (this.RenderActions[layer][i].ActionType ==
      SKRenderAction.ActionTypes.RasterImage)
00912                             {
00913                                 (SKBitmap image, bool interpolate) =
      this.Images[this.RenderActions[layer][i].ImageId];
00914
00915                                 SKPaint paint;
00916
00917                                 if (!interpolate)
00918                                 {
00919                                     paint = null;
00920                                 }
00921                                 else
00922                                 {
00923                                     paint = new SKPaint() { FilterQuality = SKFilterQuality.Medium };
00924                                 }
00925
00926                                 canvas.DrawBitmap(image,
      this.RenderActions[layer][i].ImageSource.Value, this.RenderActions[layer][i].ImageDestination.Value,
      paint);
00927
00928                                 paint?.Dispose();
00929                             }
00930                         }
00931                     }
00932
00933                     canvas.Restore();
00934                 }
00935             }
00936         }
00937
00938         private bool IsDirty = false;
00939
00940 /// <summary>
00941 /// Invalidate the contents of the canvas, forcing it to redraw itself.
00942 /// </summary>
00943         public void InvalidateDirty()
00944         {
00945             this.IsDirty = true;
00946             base.InvalidateVisual();
00947         }
00948
00949 /// <summary>
00950 /// Invalidate the contents of the canvas, specifying that the order of the layers has changed.
00951 /// </summary>
00952         public void InvalidateZIndex()
00953         {
00954             for (int i = 0; i < this.TaggedRenderActions.Count; i++)
00955             {
00956                 this.TaggedRenderActions[i] = this.TaggedRenderActions[i].OrderByDescending(a =>
      a.ZIndex).ToList();
00957             }
00958             this.InvalidateDirty();
00959         }
00960
00961
00962         private SkiaBitmap FrontBuffer = null;
00963         private RenderingParameters FrontBufferRenderingParams = null;
00964         private readonly object EmptyFrontBufferLock = new object();
00965         private object FrontBufferLock => FrontBuffer?.Lock ??  EmptyFrontBufferLock;
00966         SKCanvas FrontBufferSkiaCanvas = null;
00967
00968         private SkiaBitmap BackBuffer = null;
00969         private RenderingParameters BackBufferRenderingParams = null;
00970         SKCanvas BackBufferSkiaCanvas = null;
00971
00972         private SkiaBitmap BackBuffer2 = null;
00973         private RenderingParameters BackBufferRenderingParams2 = null;
00974         SKCanvas BackBufferSkiaCanvas2 = null;
00975
00976 /// <inheritdoc/>
```

```
00977         public override void Render(DrawingContext context)
00978         {
00979             lock (FrontBufferLock)
00980             {
00981                 double scale;
00982 
00983                 PixelPoint layoutTopLeft = this.PointToScreen(new Avalonia.Point(0, 0));
00984                 PixelPoint layoutBottomRight = this.PointToScreen(new
      Avalonia.Point(this.Bounds.Width, this.Bounds.Height));
00985 
00986                 scale = 0.5 * (this.PageWidth / (layoutBottomRight.X - layoutTopLeft.X) +
      this.PageHeight / (layoutBottomRight.Y - layoutTopLeft.Y));
00987 
00988                 Avalonia.Controls.Control parent = this.Parent as Avalonia.Controls.Control;
00989 
00990                 double left;
00991                 double top;
00992                 double width;
00993                 double height;
00994 
00995                 if (this.ClipToBounds && parent != null)
00996                 {
00997                     Avalonia.Point clipTopLeft = this.PointToClient(parent.PointToScreen(new
      Avalonia.Point(0, 0)));
00998                     Avalonia.Point clipBottomRight = this.PointToClient(parent.PointToScreen(new
      Avalonia.Point(parent.Bounds.Width, parent.Bounds.Height)));
00999 
01000                     Rect clipBounds;
01001 
01002                     if (ClipMargin.Unit == RelativeUnit.Absolute)
01003                     {
01004                         clipBounds = new Rect(clipTopLeft.X - ClipMargin.Point.X, clipTopLeft.Y -
      ClipMargin.Point.Y, clipBottomRight.X - clipTopLeft.X + ClipMargin.Point.X * 2, clipBottomRight.Y -
      clipTopLeft.Y + ClipMargin.Point.Y * 2);
01005                     }
01006                     else
01007                     {
01008                         double clipW = clipBottomRight.X - clipTopLeft.X;
01009                         double clipH = clipBottomRight.Y - clipTopLeft.Y;
01010 
01011                         clipBounds = new Rect(clipTopLeft.X - clipW * ClipMargin.Point.X,
      clipTopLeft.Y - clipH * ClipMargin.Point.Y, clipW * (1 + 2 * ClipMargin.Point.X), clipH * (1 + 2 *
      ClipMargin.Point.Y));
01012                     }
01013 
01014                     left = Math.Max(0, clipBounds.Left);
01015                     top = Math.Max(0, clipBounds.Top);
01016 
01017                     width = Math.Min(clipBounds.Width, this.PageWidth);
01018                     height = Math.Min(clipBounds.Height, this.PageHeight);
01019                 }
01020                 else
01021                 {
01022                     left = 0;
01023                     top = 0;
01024 
01025                     width = this.PageWidth;
01026                     height = this.PageHeight;
01027                 }
01028 
01029                 int pixelWidth = (int)Math.Round(width / scale);
01030                 int pixelHeight = (int)Math.Round(height / scale);
01031 
01032                 if (pixelWidth > 0 && pixelHeight > 0)
01033                 {
01034                     if (!this.ClipToBounds && (pixelWidth * pixelHeight >= int.MaxValue / 8 ||
      pixelWidth * pixelHeight < 0))
01035                     {
01036                         double ratio = width / height;
01037 
01038                         pixelHeight = (int)Math.Round(Math.Sqrt(int.MaxValue / (8 * ratio)));
01039                         pixelWidth = (int)Math.Round(pixelHeight * ratio);
01040                     }
01041 
01042                     PixelPoint topLeftScreen = this.PointToScreen(new Avalonia.Point(left, top));
01043                     Avalonia.Point topLeft = this.PointToClient(topLeftScreen);
01044                     Avalonia.Point bottomRight = this.PointToClient(new PixelPoint(topLeftScreen.X +
      pixelWidth, topLeftScreen.Y + pixelHeight));
01045 
01046                     Rect targetRect = new Rect(topLeft, bottomRight);
01047 
01048                     pixelWidth = this.PointToScreen(targetRect.BottomRight).X -
      this.PointToScreen(targetRect.TopLeft).X;
01049                     pixelHeight = this.PointToScreen(targetRect.BottomRight).Y -
      this.PointToScreen(targetRect.TopLeft).Y;
01050 
01051                     RenderingParameters currentParameters = new RenderingParameters((float)left,
```

```
      (float)top, (float)width, (float)height, (float)(1 / scale), pixelWidth, pixelHeight);
01052
01053                     if (FrontBuffer != null && FrontBufferRenderingParams == currentParameters &&
      !IsDirty)
01054                     {
01055                         if (RenderRequestedHandle.WaitOne(0))
01056                         {
01057                             Avalonia.Threading.Dispatcher.UIThread.InvokeAsync(() =>
01058                             {
01059                                 this.InvalidateVisual();
01060                             });
01061                         }
01062
01063                         context.DrawImage(FrontBuffer, new Rect(0, 0, pixelWidth, pixelHeight),
      targetRect);
01064                     }
01065                     else if (FrontBuffer != null &&
      FrontBufferRenderingParams.GoodEnough(currentParameters) && !IsDirty)
01066                     {
01067                         if (RenderRequestedHandle.WaitOne(0))
01068                         {
01069                             Avalonia.Threading.Dispatcher.UIThread.InvokeAsync(() =>
01070                             {
01071                                 this.InvalidateVisual();
01072                             });
01073                         }
01074                         context.DrawImage(FrontBuffer, new Rect(0, 0,
      FrontBufferRenderingParams.RenderWidth, FrontBufferRenderingParams.RenderHeight), new
      Rect(FrontBufferRenderingParams.Left, FrontBufferRenderingParams.Top,
      FrontBufferRenderingParams.Width, FrontBufferRenderingParams.Height));
01075                     }
01076                     else
01077                     {
01078                         lock (RenderingRequestLock)
01079                         {
01080                             IsDirty = false;
01081                             RenderingRequest = currentParameters;
01082                             RenderRequestedHandle.Set();
01083                         }
01084
01085                         if (FrontBuffer != null)
01086                         {
01087                             context.DrawImage(FrontBuffer, new Rect(0, 0,
      FrontBufferRenderingParams.RenderWidth, FrontBufferRenderingParams.RenderHeight), new
      Rect(FrontBufferRenderingParams.Left, FrontBufferRenderingParams.Top,
      FrontBufferRenderingParams.Width, FrontBufferRenderingParams.Height));
01088                         }
01089
01090                         Avalonia.Threading.Dispatcher.UIThread.InvokeAsync(() =>
01091                         {
01092                             this.InvalidateVisual();
01093                         });
01094                     }
01095                 }
01096             }
01097         }
01098
01099         private bool disposedValue;
01100
01101 /// <inheritdoc cref="Dispose()"/>
01102         protected virtual void Dispose(bool disposing)
01103         {
01104             if (!disposedValue)
01105             {
01106                 if (disposing)
01107                 {
01108                     this.BackBuffer?.Dispose();
01109                     this.BackBuffer2?.Dispose();
01110                     this.DisposedHandle?.Set();
01111                     this.FrontBuffer?.Dispose();
01112
01113                     foreach (KeyValuePair<string, (SKBitmap, bool)> image in this.Images)
01114                     {
01115                         image.Value.Item1?.Dispose();
01116                     }
01117
01118                     for (int i = 0; i < this.RenderActions?.Count; i++)
01119                     {
01120                         foreach (SKRenderAction act in this.RenderActions[i])
01121                         {
01122                             act?.Dispose();
01123                         }
01124
01125                         this.LayerTransforms?[i]?.Dispose();
01126                     }
01127                 }
01128
```

```
01129                 disposedValue = true;
01130             }
01131         }
01132 
01133 /// <inheritdoc/>
01134         public void Dispose()
01135         {
01136             Dispose(disposing:  true);
01137             GC.SuppressFinalize(this);
01138         }
01139     }
01140 }
```

8.6 SKRenderContext.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2023 Giorgio Bianchini, University of Bristol
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using SkiaSharp;
00019 using System;
00020 using System.Collections.Generic;
00021 using System.Linq;
00022 using VectSharp.Filters;
00023 
00024 namespace VectSharp.Canvas
00025 {
00026 
00027 /// <summary>
00028 /// Represents a light-weight rendering action.
00029 /// </summary>
00030     public class SKRenderAction :  IDisposable
00031     {
00032 /// <summary>
00033 /// Returns a boolean value indicating whether the current instance has been disposed.
00034 /// </summary>
00035         public bool Disposed => disposedValue;
00036 
00037         private bool disposedValue;
00038 
00039 /// <summary>
00040 /// Types of rendering actions.
00041 /// </summary>
00042         public enum ActionTypes
00043         {
00044 /// <summary>
00045 /// The render action represents a path object.
00046 /// </summary>
00047             Path,
00048 
00049 /// <summary>
00050 /// The render action represents a text object.
00051 /// </summary>
00052             Text,
00053 
00054 /// <summary>
00055 /// The render action represents a raster image.
00056 /// </summary>
00057             RasterImage,
00058 
00059 /// <summary>
00060 /// The render action represents a transformation of the coordinate space.
00061 /// </summary>
00062             Transform,
00063 
00064 /// <summary>
00065 /// The render action represents saving the current graphics state.
00066 /// </summary>
00067             Save,
00068 
```

```
00069 /// <summary>
00070 /// The render action represents restoring the last saved graphics state.
00071 /// </summary>
00072             Restore,
00073 
00074 /// <summary>
00075 /// The render action represents an update of the current clip path.
00076 /// </summary>
00077             Clip,
00078 
00079 /// <summary>
00080 /// The render action represents rendering a graphics object with a filter.
00081 /// </summary>
00082             DrawFiltered
00083         }
00084 
00085 /// <summary>
00086 /// Type of the rendering action.
00087 /// </summary>
00088         public ActionTypes ActionType { get; private set; }
00089 
00090 /// <summary>
00091 /// Path that needs to be rendered (null if the action type is not <see cref="ActionTypes.Path"/>).
      If you change this, you probably want to call this object's <see cref="InvalidateHitTestPath"/>
      method.
00092 /// </summary>
00093         public SKPath Path { get; set; }
00094 
00095         internal SKPath HitTestPath { get; set; }
00096         internal SKPath LastRenderedGlobalHitTestPath { get; set; }
00097 
00098 /// <summary>
00099 /// Text that needs to be rendered (null if the action type is not <see cref="ActionTypes.Text"/>).
      If you change this, you probably want to call this object's <see cref="InvalidateHitTestPath"/>
      method.
00100 /// </summary>
00101         public string Text { get; set; }
00102 
00103 /// <summary>
00104 /// The font that will be used to render the text (null if the action type is not <see
      cref="ActionTypes.Text"/>).  If you change this, you probably want to call this object's <see
      cref="InvalidateHitTestPath"/> method.
00105 /// </summary>
00106         public SKFont Font { get; set; }
00107 
00108 /// <summary>
00109 /// The X coordainate at which the text will be drawn (null if the action type is not <see
      cref="ActionTypes.Text"/>).  If you change this, you probably want to call this object's <see
      cref="InvalidateHitTestPath"/> method.
00110 /// </summary>
00111         public float TextX { get; set; }
00112 
00113 /// <summary>
00114 /// The Y coordainate at which the text will be drawn (null if the action type is not <see
      cref="ActionTypes.Text"/>).  If you change this, you probably want to call this object's <see
      cref="InvalidateHitTestPath"/> method.
00115 /// </summary>
00116         public float TextY { get; set; }
00117 
00118 /// <summary>
00119 /// Paint used to render the text or path (<see langword="null"/> if the action type is neither <see
      cref="ActionTypes.Text"/> nor <see cref="ActionTypes.Path"/>).  If you change this, you probably want
      to call this object's <see cref="InvalidateHitTestPath"/> method.
00120 /// </summary>
00121         public SKPaint Paint { get; set; }
00122 
00123 /// <summary>
00124 /// Univocal identifier of the image that needs to be drawn.
00125 /// </summary>
00126         public string ImageId { get; set; }
00127 
00128 /// <summary>
00129 /// The source rectangle of the image (<see langword="null"/> if the action type is not <see
      cref="ActionTypes.RasterImage"/>).  If you change this, you probably want to call this object's <see
      cref="InvalidateVisual"/> method.
00130 /// </summary>
00131         public SKRect?  ImageSource { get; set; }
00132 
00133 /// <summary>
00134 /// The destination rectangle of the image (<see langword="null"/> if the action type is not <see
      cref="ActionTypes.RasterImage"/>).  If you change this, you probably want to call this object's <see
      cref="InvalidateHitTestPath"/> method.
00135 /// </summary>
00136         public SKRect?  ImageDestination { get; set; }
00137 
00138 /// <summary>
00139 /// The transformation matrix that will be applied to the current coordinate system (<see
```

```
langword="null"/> if the action type is not <see cref="ActionTypes.Transform"/>).  If you change this, you probably want to call this object's <see cref="InvalidateVisual"/> method.
00140 /// </summary>
00141         public SKMatrix?  Transform { get; set; } = null;
00142 
00143 /// <summary>
00144 /// A tag to access the <see cref="SKRenderAction"/>.
00145 /// </summary>
00146         public string Tag { get; private set; }
00147 
00148 /// <summary>
00149 /// The Z-index of the rendering action (an action with a higher Z-index will always appear above an action with a lower Z-index).
00150 /// The more different values there are for the Z-index, the slower the rendering, so keep use of this property to a minimum.
00151 /// If you change this, you probably want to call this object's <see cref="InvalidateZIndex"/> method.
00152 /// </summary>
00153         public uint ZIndex { get; set; } = 0;
00154 
00155 /// <summary>
00156 /// The graphics that will be drawn with the specified <see cref="Filter"/>.  If you change this, you probably want to call this object's <see cref="InvalidateVisual"/> method.
00157 /// </summary>
00158         public SKRenderContext Graphics { get; set; } = null;
00159 
00160 /// <summary>
00161 /// The filter with which the <see cref="Graphics"/> is drawn.  If you change this, you probably want to call this object's <see cref="InvalidateVisual"/> method.
00162 /// </summary>
00163         public IFilter Filter { get; set; }
00164 
00165 /// <summary>
00166 /// An arbitrary object associated with the RenderAction.
00167 /// </summary>
00168         public object Payload { get; set; }
00169 
00170         internal ISKRenderCanvas InternalParent { get; set; }
00171 
00172 /// <summary>
00173 /// The container of this <see cref="SKRenderAction"/>.
00174 /// </summary>
00175         public Avalonia.Controls.Canvas Parent
00176         {
00177             get
00178             {
00179                 return (Avalonia.Controls.Canvas)InternalParent;
00180             }
00181         }
00182 
00183 /// <summary>
00184 /// Raised when the pointer enters the area covered by the <see cref="SKRenderAction"/>.
00185 /// </summary>
00186         public event EventHandler<Avalonia.Input.PointerEventArgs> PointerEntered;
00187 
00188 /// <summary>
00189 /// Raised when the pointer leaves the area covered by the <see cref="SKRenderAction"/>.
00190 /// </summary>
00191         public event EventHandler<Avalonia.Input.PointerEventArgs> PointerExited;
00192 
00193 /// <summary>
00194 /// Raised when the pointer is pressed while over the area covered by the <see cref="SKRenderAction"/>.
00195 /// </summary>
00196         public event EventHandler<Avalonia.Input.PointerPressedEventArgs> PointerPressed;
00197 
00198 /// <summary>
00199 /// Raised when the pointer is released after a <see cref="PointerPressed"/> event.
00200 /// </summary>
00201         public event EventHandler<Avalonia.Input.PointerReleasedEventArgs> PointerReleased;
00202 
00203 
00204         internal void FirePointerEntered(Avalonia.Input.PointerEventArgs e)
00205         {
00206             this.PointerEntered?.Invoke(this, e);
00207         }
00208 
00209         internal void FirePointerExited(Avalonia.Input.PointerEventArgs e)
00210         {
00211             this.PointerExited?.Invoke(this, e);
00212         }
00213 
00214         internal void FirePointerPressed(Avalonia.Input.PointerPressedEventArgs e)
00215         {
00216             this.PointerPressed?.Invoke(this, e);
00217         }
00218 
00219         internal void FirePointerReleased(Avalonia.Input.PointerReleasedEventArgs e)
```

```
00220         {
00221             this.PointerReleased?.Invoke(this, e);
00222         }
00223 
00224 /// <summary>
00225 /// <para>Signals to this object that its shape has changed and a new path needs to be computed for
      the purpose of hit-testing.
00226 /// Also signals to the <see cref="Parent"/> that the visual properties of this object have changed
      and triggers a redraw.</para>
00227 /// <para>This method should be called whenever the "shape" of the object represented by the <see
      cref="SKRenderAction"/> changes.
00228 /// If only the visual properties of this object have changed (e.g.  the colour), call the <see
      cref="InvalidateVisual"/> method instead.</para>
00229 /// <para>If you make changes to more than one <see cref="SKRenderAction"/> contained in the same <see
      cref="SKMultiLayerRenderCanvas"/>, you only need to invalidate the last one.</para>
00230 /// <para>This method should only be called after the output has been fully initialized.</para>
00231 /// </summary>
00232         public void InvalidateHitTestPath()
00233         {
00234             CreateHitTestPath();
00235 
00236             this.InvalidateVisual();
00237         }
00238 
00239         internal void CreateHitTestPath()
00240         {
00241             if (this.ActionType == ActionTypes.Path)
00242             {
00243                 if (this.Path != null && this.Paint != null)
00244                 {
00245                     this.HitTestPath?.Dispose();
00246                     this.HitTestPath = this.Paint.GetFillPath(this.Path);
00247                 }
00248             }
00249             else if (this.ActionType == ActionTypes.RasterImage)
00250             {
00251                 if (this.ImageDestination != null)
00252                 {
00253                     this.HitTestPath?.Dispose();
00254                     this.HitTestPath = new SKPath();
00255                     this.HitTestPath.AddRect(this.ImageDestination.Value);
00256                 }
00257             }
00258             else if (this.ActionType == ActionTypes.Text)
00259             {
00260                 if (this.Paint != null && this.Font != null)
00261                 {
00262                     this.HitTestPath?.Dispose();
00263                     this.Paint.Typeface = this.Font.Typeface;
00264                     this.Paint.TextSize = this.Font.Size;
00265 
00266                     using (SKPath pth = this.Paint.GetTextPath(this.Text, this.TextX, this.TextY))
00267                     {
00268                         this.HitTestPath = this.Paint.GetFillPath(pth);
00269                     }
00270                 }
00271             }
00272         }
00273 
00274 /// <summary>
00275 /// <para>This methods signals to the <see cref="Parent"/> that the visual properties (e.g.  the
      colour) of this object have changed and triggers a redraw.</para>
00276 /// <para>If the "shape" of the object has changed as well, call the <see
      cref="InvalidateHitTestPath"/> method instead.  If the Z-index of the
00277 /// object has changed, call the <see cref="InvalidateZIndex"/> method instead.  If both the "shape"
      and the Z-index of the object have changed,
00278 /// call the <see cref="InvalidateAll"/> method.</para>
00279 /// <para>If you make changes to more than one <see cref="SKRenderAction"/> contained in the same <see
      cref="Canvas"/>, you only need to invalidate the last one.</para>
00280 /// <para>This method should only be called after the output has been fully initialized.</para>
00281 /// </summary>
00282         public void InvalidateVisual()
00283         {
00284             this.InternalParent?.InvalidateDirty();
00285         }
00286 
00287 /// <summary>
00288 /// <para>This methods signals to the <see cref="Parent"/> that the Z-index and visual properties
      (e.g.  the colour) of this object have changed and triggers a redraw.</para>
00289 /// <para>If the "shape" of the object has changed as well, call the <see cref="InvalidateAll"/>
      method instead.</para>
00290 /// <para>If you make changes to more than one <see cref="SKRenderAction"/> contained in the same <see
      cref="Canvas"/>, you only need to invalidate the last one.</para>
00291 /// <para>This method should only be called after the output has been fully initialized.</para>
00292 /// </summary>
00293         public void InvalidateZIndex()
00294         {
```

```
00295             this.InternalParent?.InvalidateZIndex();
00296         }
00297 
00298 /// <summary>
00299 /// <para>This methods signals to the <see cref="Parent"/> that the Z-index, shape and visual
      properties (e.g.  the colour) of this object have changed and triggers a redraw.</para>
00300 /// <para>If you make changes to more than one <see cref="SKRenderAction"/> contained in the same <see
      cref="Canvas"/>, you only need to invalidate the last one.</para>
00301 /// <para>This method should only be called after the output has been fully initialized.</para>
00302 /// </summary>
00303         public void InvalidateAll()
00304         {
00305             this.InvalidateHitTestPath();
00306             this.InvalidateZIndex();
00307         }
00308 
00309         private SKRenderAction()
00310         {
00311 
00312         }
00313 
00314 /// <summary>
00315 /// Creates a new <see cref="SKRenderAction"/> representing a path.
00316 /// </summary>
00317 /// <param name="path">The geometry to be rendered.</param>
00318 /// <param name="paint">The paint used to fill or stroke the path.</param>
00319 /// <param name="tag">A tag to access the <see cref="SKRenderAction"/>.  If this is null this <see
      cref="SKRenderAction"/> is not visible in the hit test.</param>
00320 /// <returns>A new <see cref="SKRenderAction"/> representing a path.</returns>
00321         public static SKRenderAction PathAction(SKPath path, SKPaint paint, string tag = null)
00322         {
00323             SKRenderAction act = new SKRenderAction()
00324             {
00325                 ActionType = ActionTypes.Path,
00326                 Path = path,
00327                 Paint = paint,
00328                 Tag = tag
00329             };
00330 
00331             if (!string.IsNullOrEmpty(tag))
00332             {
00333                 act.CreateHitTestPath();
00334             }
00335 
00336             return act;
00337         }
00338 
00339 /// <summary>
00340 /// Creates a new <see cref="SKRenderAction"/> representing a clipping action.
00341 /// </summary>
00342 /// <param name="clippingPath">The path to be used for clipping.</param>
00343 /// <param name="tag">A tag to access the <see cref="SKRenderAction"/>.</param>
00344 /// <returns>A new <see cref="SKRenderAction"/> representing a clipping action.</returns>
00345         public static SKRenderAction ClipAction(SKPath clippingPath, string tag = null)
00346         {
00347             return new SKRenderAction()
00348             {
00349                 ActionType = ActionTypes.Clip,
00350                 Path = clippingPath,
00351                 Tag = tag
00352             };
00353         }
00354 
00355 /// <summary>
00356 /// Creates a new <see cref="SKRenderAction"/> representing text.
00357 /// </summary>
00358 /// <param name="text">The text to be rendered.</param>
00359 /// <param name="x">The X coordinate at which the text will be drawn.</param>
00360 /// <param name="y">The Y coordinate at which the text will be drawn.</param>
00361 /// <param name="font">The font to be used to render the text.</param>
00362 /// <param name="paint">The paint to be used to fill or stroke the text.</param>
00363 /// <param name="tag">A tag to access the <see cref="SKRenderAction"/>.  If this is null this <see
      cref="SKRenderAction"/> is not visible in the hit test.</param>
00364 /// <returns>A new <see cref="SKRenderAction"/> representing text.</returns>
00365         public static SKRenderAction TextAction(string text, float x, float y, SKFont font, SKPaint
      paint, string tag = null)
00366         {
00367             SKRenderAction act = new SKRenderAction()
00368             {
00369                 ActionType = ActionTypes.Text,
00370                 Text = text,
00371                 TextX = x,
00372                 TextY = y,
00373                 Font = font,
00374                 Paint = paint,
00375                 Tag = tag
00376             };
```

```
00377
00378             act.Paint.Typeface = font.Typeface;
00379             act.Paint.TextSize = font.Size;
00380
00381             if (!string.IsNullOrEmpty(tag))
00382             {
00383                 act.CreateHitTestPath();
00384             }
00385
00386             return act;
00387         }
00388
00389 /// <summary>
00390 /// Creates a new <see cref="SKRenderAction"/> representing an image.
00391 /// </summary>
00392 /// <param name="imageId">The univocal identifier of the image to draw.</param>
00393 /// <param name="sourceRect">The source rectangle of the image.</param>
00394 /// <param name="destinationRect">The destination rectangle of the image.</param>
00395 /// <param name="tag">A tag to access the <see cref="SKRenderAction"/>.  If this is null this <see
      cref="SKRenderAction"/> is not visible in the hit test.</param>
00396 /// <returns>A new <see cref="SKRenderAction"/> representing an image.</returns>
00397         public static SKRenderAction ImageAction(string imageId, SKRect sourceRect, SKRect
      destinationRect, string tag = null)
00398         {
00399             SKRenderAction act = new SKRenderAction()
00400             {
00401                 ActionType = ActionTypes.RasterImage,
00402                 ImageId = imageId,
00403                 ImageSource = sourceRect,
00404                 ImageDestination = destinationRect,
00405                 Tag = tag
00406             };
00407
00408             if (!string.IsNullOrEmpty(tag))
00409             {
00410                 act.CreateHitTestPath();
00411             }
00412
00413             return act;
00414         }
00415
00416 /// <summary>
00417 /// Creates a new <see cref="SKRenderAction"/> representing a transform.
00418 /// </summary>
00419 /// <param name="transform">The transform to apply.</param>
00420 /// <param name="tag">A tag to access the <see cref="SKRenderAction"/>.</param>
00421 /// <returns>A new <see cref="SKRenderAction"/> representing a transform.</returns>
00422         public static SKRenderAction TransformAction(SKMatrix transform, string tag = null)
00423         {
00424             return new SKRenderAction()
00425             {
00426                 ActionType = ActionTypes.Transform,
00427                 Transform = transform,
00428                 Tag = tag
00429             };
00430         }
00431
00432 /// <summary>
00433 /// Creates a new <see cref="SKRenderAction"/> that saves the current graphics state.
00434 /// </summary>
00435 /// <param name="tag">A tag to access the <see cref="SKRenderAction"/>.</param>
00436 /// <returns>A new <see cref="SKRenderAction"/> that saves the current graphics state.</returns>
00437         public static SKRenderAction SaveAction(string tag = null)
00438         {
00439             return new SKRenderAction()
00440             {
00441                 ActionType = ActionTypes.Save,
00442                 Tag = tag
00443             };
00444         }
00445
00446 /// <summary>
00447 /// Creates a new <see cref="SKRenderAction"/> that saves the current graphics state.
00448 /// </summary>
00449 /// <param name="tag">A tag to access the <see cref="SKRenderAction"/>.</param>
00450 /// <returns>A new <see cref="SKRenderAction"/> that restores the last saved graphics state.</returns>
00451         public static SKRenderAction RestoreAction(string tag = null)
00452         {
00453             return new SKRenderAction()
00454             {
00455                 ActionType = ActionTypes.Restore,
00456                 Tag = tag
00457             };
00458         }
00459
00460 /// <summary>
00461 /// Create a new <see cref="SKRenderAction"/> that draws some graphics with a filter.
```

```
00462 /// </summary>
00463 /// <returns>A new <see cref="SKRenderAction"/> that draws some graphics with a filter.</returns>
00464         public static SKRenderAction DrawFilteredGraphicsAction(SKRenderContext graphics, IFilter
      filter, string tag = null)
00465         {
00466             return new SKRenderAction()
00467             {
00468                 ActionType = ActionTypes.DrawFiltered,
00469                 Graphics = graphics,
00470                 Filter = filter,
00471                 Tag = tag
00472             };
00473         }
00474 
00475 /// <inheritdoc cref="IDisposable.Dispose"/>
00476         protected virtual void Dispose(bool disposing)
00477         {
00478             if (!disposedValue)
00479             {
00480                 if (disposing)
00481                 {
00482                     this.Font?.Dispose();
00483                     this.HitTestPath?.Dispose();
00484                     this.LastRenderedGlobalHitTestPath?.Dispose();
00485                     this.Paint?.Dispose();
00486                     this.Path?.Dispose();
00487                 }
00488 
00489                 disposedValue = true;
00490             }
00491         }
00492 
00493 /// <inheritdoc cref="IDisposable.Dispose"/>
00494         public void Dispose()
00495         {
00496             Dispose(disposing:  true);
00497             GC.SuppressFinalize(this);
00498         }
00499     }
00500 
00501 
00502     internal static class SKTypefaceCache
00503     {
00504         private static readonly object LockObject = new object();
00505         private static readonly Dictionary<string, SKTypeface> Typefaces = new Dictionary<string,
      SKTypeface>();
00506         public static SKTypeface GetSKTypeface(FontFamily family)
00507         {
00508 
00509             lock (LockObject)
00510             {
00511                 if (Typefaces.TryGetValue(family.FileName, out SKTypeface tbr))
00512                 {
00513                     return tbr;
00514                 }
00515                 else
00516                 {
00517                     try
00518                     {
00519                         System.IO.MemoryStream fontStream = new
      System.IO.MemoryStream((int)family.TrueTypeFile.FontStream.Length);
00520                         family.TrueTypeFile.FontStream.Seek(0, System.IO.SeekOrigin.Begin);
00521                         family.TrueTypeFile.FontStream.CopyTo(fontStream);
00522                         fontStream.Seek(0, System.IO.SeekOrigin.Begin);
00523 
00524                         SKTypeface typeface = SKTypeface.FromData(SKData.Create(fontStream));
00525 
00526                         Typefaces[family.FileName] = typeface;
00527 
00528                         return typeface;
00529                     }
00530                     catch
00531                     {
00532                         SKTypeface typeface = SKTypeface.Default;
00533 
00534                         Typefaces[family.FileName] = typeface;
00535 
00536                         return typeface;
00537                     }
00538                 }
00539             }
00540         }
00541     }
00542 
00543 /// <summary>
00544 /// Represents a page that has been prepared for fast rendering using the SkiaSharp renderer.
00545 /// </summary>
```

```
00546     public class SKRenderContext
00547     {
00548         internal virtual Dictionary<string, (SKBitmap, bool)> Images { get; set; }
00549         internal virtual List<SKRenderAction> SKRenderActions { get; set; }
00550     }
00551
00552
00553     internal class SKRenderContextImpl :  SKRenderContext, IGraphicsContext, IDisposable
00554     {
00555         public Dictionary<string, Func<SKRenderAction, IEnumerable<SKRenderAction>> TaggedActions {
      get; set; } = new Dictionary<string, Func<SKRenderAction, IEnumerable<SKRenderAction>>();
00556
00557         private readonly bool removeTaggedActions = true;
00558
00559         public string Tag { get; set; }
00560
00561         private readonly AvaloniaContextInterpreter.TextOptions _textOption;
00562
00563         internal override Dictionary<string, (SKBitmap, bool)> Images { get; set; }
00564
00565         private readonly FilterOption _filterOption;
00566
00567         public SKRenderContextImpl(double width, double height, bool
      removeTaggedActionsAfterExecution, AvaloniaContextInterpreter.TextOptions textOption,
      Dictionary<string, (SKBitmap, bool)> images, FilterOption filterOption)
00568         {
00569             this.Images = images;
00570
00571             currentPath = null;
00572             figureInitialised = false;
00573
00574             SKRenderActions = new List<SKRenderAction>();
00575             removeTaggedActions = removeTaggedActionsAfterExecution;
00576
00577             Width = width;
00578             Height = height;
00579
00580             _textOption = textOption;
00581             _filterOption = filterOption;
00582         }
00583
00584         internal override List<SKRenderAction> SKRenderActions { get; set; }
00585
00586         public double Width { get; private set; }
00587         public double Height { get; private set; }
00588
00589         private void AddAction(SKRenderAction act)
00590         {
00591             if (!string.IsNullOrEmpty(Tag))
00592             {
00593                 if (TaggedActions.ContainsKey(Tag))
00594                 {
00595                     IEnumerable<SKRenderAction> actions = TaggedActions[Tag](act);
00596
00597                     foreach (SKRenderAction action in actions)
00598                     {
00599                         SKRenderActions.Add(action);
00600                     }
00601
00602                     if (removeTaggedActions)
00603                     {
00604                         TaggedActions.Remove(Tag);
00605                     }
00606                 }
00607                 else
00608                 {
00609                     SKRenderActions.Add(act);
00610                 }
00611             }
00612             else if (TaggedActions.ContainsKey(""))
00613             {
00614                 IEnumerable<SKRenderAction> actions = TaggedActions[""](act);
00615
00616                 foreach (SKRenderAction action in actions)
00617                 {
00618                     SKRenderActions.Add(action);
00619                 }
00620             }
00621             else
00622             {
00623                 SKRenderActions.Add(act);
00624             }
00625         }
00626
00627         public void Translate(double x, double y)
00628         {
00629             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
```

```
00630
00631             SKRenderAction act = SKRenderAction.TransformAction(SKMatrix.CreateTranslation((float)x,
      (float)y), Tag);
00632             AddAction(act);
00633
00634             currentPath = null;
00635             figureInitialised = false;
00636         }
00637
00638         public TextBaselines TextBaseline { get; set; }
00639
00640         private void PathText(string text, double x, double y)
00641         {
00642             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00643
00644             GraphicsPath textPath = new GraphicsPath().AddText(x, y, text, Font, TextBaseline);
00645
00646             for (int j = 0; j < textPath.Segments.Count; j++)
00647             {
00648                 switch (textPath.Segments[j].Type)
00649                 {
00650                     case VectSharp.SegmentType.Move:
00651                         this.MoveTo(textPath.Segments[j].Point.X, textPath.Segments[j].Point.Y);
00652                         break;
00653                     case VectSharp.SegmentType.Line:
00654                         this.LineTo(textPath.Segments[j].Point.X, textPath.Segments[j].Point.Y);
00655                         break;
00656                     case VectSharp.SegmentType.CubicBezier:
00657                         this.CubicBezierTo(textPath.Segments[j].Points[0].X,
      textPath.Segments[j].Points[0].Y, textPath.Segments[j].Points[1].X, textPath.Segments[j].Points[1].Y,
      textPath.Segments[j].Points[2].X, textPath.Segments[j].Points[2].Y);
00658                         break;
00659                     case VectSharp.SegmentType.Close:
00660                         this.Close();
00661                         break;
00662                 }
00663             }
00664         }
00665
00666         public void StrokeSimpleText(string text, double x, double y)
00667         {
00668             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00669
00670             if ((_textOption == AvaloniaContextInterpreter.TextOptions.NeverConvert || (_textOption ==
      AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary && (Font.FontFamily.IsStandardFamily ||
      Font.FontFamily.TrueTypeFile?.FontStream != null))) &&
      !System.Runtime.InteropServices.RuntimeInformation.IsOSPlatform(System.Runtime.InteropServices.OSPlatform.Linux))
00671             {
00672                 SKTypeface typeface = SKTypefaceCache.GetSKTypeface(Font.FontFamily);
00673
00674                 double top = y;
00675                 double left = x;
00676
00677                 Font.DetailedFontMetrics metrics = Font.MeasureTextAdvanced(text);
00678
00679                 if (TextBaseline == TextBaselines.Top)
00680                 {
00681                     if (Font.FontFamily.TrueTypeFile != null)
00682                     {
00683                         left -= metrics.LeftSideBearing;
00684                         top += metrics.Top;
00685                     }
00686                 }
00687                 else if (TextBaseline == TextBaselines.Middle)
00688                 {
00689                     if (Font.FontFamily.TrueTypeFile != null)
00690                     {
00691                         left -= metrics.LeftSideBearing;
00692                         top += (metrics.Top + metrics.Bottom) * 0.5;
00693                     }
00694                 }
00695                 else if (TextBaseline == TextBaselines.Baseline)
00696                 {
00697                     if (Font.FontFamily.TrueTypeFile != null)
00698                     {
00699                         left -= metrics.LeftSideBearing;
00700                     }
00701                 }
00702                 else if (TextBaseline == TextBaselines.Bottom)
00703                 {
00704                     if (Font.FontFamily.TrueTypeFile != null)
00705                     {
00706                         left -= metrics.LeftSideBearing;
00707                         top += metrics.Bottom;
00708                     }
00709                 }
00710
```

```
00711                 SKPaint stroke = new SKPaint() { IsStroke = true, IsAntialias = true, Style =
      SKPaintStyle.Stroke, StrokeWidth = (float)LineWidth, SubpixelText = true };
00712
00713                 if (this.StrokeStyle is SolidColourBrush solid)
00714                 {
00715                     stroke.Color = new SKColor((byte)(solid.R * 255), (byte)(solid.G * 255),
      (byte)(solid.B * 255), StrokeAlpha);
00716                 }
00717                 else if (this.StrokeStyle is LinearGradientBrush linearGradient)
00718                 {
00719                     stroke.Shader = linearGradient.ToSKShader();
00720                 }
00721                 else if (this.StrokeStyle is RadialGradientBrush radialGradient)
00722                 {
00723                     stroke.Shader = radialGradient.ToSKShader();
00724                 }
00725
00726                 stroke.PathEffect = SKPathEffect.CreateDash(LineDash.DashArray.Select(x =>
      (float)x).ToArray(), (float)LineDash.Phase);
00727
00728                 switch (LineCap)
00729                 {
00730                     case LineCaps.Butt:
00731                         stroke.StrokeCap = SKStrokeCap.Butt;
00732                         break;
00733                     case LineCaps.Round:
00734                         stroke.StrokeCap = SKStrokeCap.Round;
00735                         break;
00736                     case LineCaps.Square:
00737                         stroke.StrokeCap = SKStrokeCap.Square;
00738                         break;
00739                 }
00740
00741                 switch (LineJoin)
00742                 {
00743                     case LineJoins.Bevel:
00744                         stroke.StrokeJoin = SKStrokeJoin.Bevel;
00745                         break;
00746                     case LineJoins.Round:
00747                         stroke.StrokeJoin = SKStrokeJoin.Round;
00748                         break;
00749                     case LineJoins.Miter:
00750                         stroke.StrokeJoin = SKStrokeJoin.Miter;
00751                         break;
00752                 }
00753
00754                 SKRenderAction act = SKRenderAction.TextAction(text, (float)left, (float)top, new
      SKFont(typeface, (float)Font.FontSize), stroke, Tag);
00755
00756                 AddAction(act);
00757             }
00758             else
00759             {
00760                 PathText(text, x, y);
00761                 Stroke();
00762             }
00763         }
00764
00765         public void FillText(string text, double x, double y)
00766         {
00767             if (!Font.EnableKerning)
00768             {
00769                 FillSimpleText(text, x, y);
00770             }
00771             else
00772             {
00773                 List<(string, Point)> tSpans = new List<(string, Point)>();
00774
00775                 System.Text.StringBuilder currentRun = new System.Text.StringBuilder();
00776                 Point currentKerning = new Point();
00777
00778                 Point currentGlyphPlacementDelta = new Point();
00779                 Point currentGlyphAdvanceDelta = new Point();
00780                 Point nextGlyphPlacementDelta = new Point();
00781                 Point nextGlyphAdvanceDelta = new Point();
00782
00783                 for (int i = 0; i < text.Length; i++)
00784                 {
00785                     if (i < text.Length - 1)
00786                     {
00787                         currentGlyphPlacementDelta = nextGlyphPlacementDelta;
00788                         currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
00789                         nextGlyphAdvanceDelta = new Point();
00790                         nextGlyphPlacementDelta = new Point();
00791
00792                         TrueTypeFile.PairKerning kerning =
      Font.FontFamily.TrueTypeFile.Get1000EmKerning(text[i], text[i + 1]);
```

```
00793
00794                         if (kerning != null)
00795                         {
00796                             currentGlyphPlacementDelta = new Point(currentGlyphPlacementDelta.X +
      kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y + kerning.Glyph1Placement.Y);
00797                             currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
      kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
00798
00799                             nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
      kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
00800                             nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
      kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
00801                         }
00802                     }
00803
00804                     if (currentGlyphPlacementDelta.X != 0 || currentGlyphPlacementDelta.Y != 0 ||
      currentGlyphAdvanceDelta.X != 0 || currentGlyphAdvanceDelta.Y != 0)
00805                     {
00806                         if (currentRun.Length > 0)
00807                         {
00808                             tSpans.Add((currentRun.ToString(), currentKerning));
00809
00810                             tSpans.Add((text[i].ToString(), new Point(currentGlyphPlacementDelta.X *
      Font.FontSize / 1000, currentGlyphPlacementDelta.Y * Font.FontSize / 1000)));
00811
00812                             currentRun.Clear();
00813                             currentKerning = new Point((currentGlyphAdvanceDelta.X -
      currentGlyphPlacementDelta.X) * Font.FontSize / 1000, (currentGlyphAdvanceDelta.Y -
      currentGlyphPlacementDelta.Y) * Font.FontSize / 1000);
00814                         }
00815                         else
00816                         {
00817                             tSpans.Add((text[i].ToString(), new Point(currentGlyphPlacementDelta.X *
      Font.FontSize / 1000 + currentKerning.X, currentGlyphPlacementDelta.Y * Font.FontSize / 1000 +
      currentKerning.Y)));
00818
00819                             currentRun.Clear();
00820                             currentKerning = new Point((currentGlyphAdvanceDelta.X -
      currentGlyphPlacementDelta.X) * Font.FontSize / 1000, (currentGlyphAdvanceDelta.Y -
      currentGlyphPlacementDelta.Y) * Font.FontSize / 1000);
00821                         }
00822                     }
00823                     else
00824                     {
00825                         currentRun.Append(text[i]);
00826                     }
00827                 }
00828
00829                 if (currentRun.Length > 0)
00830                 {
00831                     tSpans.Add((currentRun.ToString(), currentKerning));
00832                 }
00833
00834                 double currX = x;
00835                 double currY = y;
00836
00837                 Font.DetailedFontMetrics fullMetrics = Font.MeasureTextAdvanced(text);
00838
00839                 if (TextBaseline == TextBaselines.Top)
00840                 {
00841                     if (Font.FontFamily.TrueTypeFile != null)
00842                     {
00843                         currY += fullMetrics.Top;
00844                     }
00845                 }
00846                 else if (TextBaseline == TextBaselines.Middle)
00847                 {
00848                     if (Font.FontFamily.TrueTypeFile != null)
00849                     {
00850                         currY += (fullMetrics.Top + fullMetrics.Bottom) * 0.5;
00851                     }
00852                 }
00853                 else if (TextBaseline == TextBaselines.Bottom)
00854                 {
00855                     if (Font.FontFamily.TrueTypeFile != null)
00856                     {
00857                         currY += fullMetrics.Bottom;
00858                     }
00859                 }
00860
00861                 TextBaseline = TextBaselines.Baseline;
00862
00863                 for (int i = 0; i < tSpans.Count; i++)
00864                 {
00865                     Font.DetailedFontMetrics metrics = Font.MeasureTextAdvanced(tSpans[i].Item1);
00866
00867                     if (i == 0)
```

```
00868                     {
00869                         FillSimpleText(tSpans[i].Item1, currX + tSpans[i].Item2.X, currY +
    tSpans[i].Item2.Y);
00870                     }
00871                     else
00872                     {
00873                         FillSimpleText(tSpans[i].Item1, currX + metrics.LeftSideBearing -
    fullMetrics.LeftSideBearing + tSpans[i].Item2.X, currY + tSpans[i].Item2.Y);
00874                     }
00875 
00876 
00877                     currX += metrics.AdvanceWidth + tSpans[i].Item2.X;
00878                     currY += tSpans[i].Item2.Y;
00879                 }
00880             }
00881         }
00882 
00883         public void StrokeText(string text, double x, double y)
00884         {
00885             if (!Font.EnableKerning)
00886             {
00887                 StrokeSimpleText(text, x, y);
00888             }
00889             else
00890             {
00891                 List<(string, Point)> tSpans = new List<(string, Point)>();
00892 
00893                 System.Text.StringBuilder currentRun = new System.Text.StringBuilder();
00894                 Point currentKerning = new Point();
00895 
00896                 Point currentGlyphPlacementDelta = new Point();
00897                 Point currentGlyphAdvanceDelta = new Point();
00898                 Point nextGlyphPlacementDelta = new Point();
00899                 Point nextGlyphAdvanceDelta = new Point();
00900 
00901                 for (int i = 0; i < text.Length; i++)
00902                 {
00903                     if (i < text.Length - 1)
00904                     {
00905                         currentGlyphPlacementDelta = nextGlyphPlacementDelta;
00906                         currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
00907                         nextGlyphAdvanceDelta = new Point();
00908                         nextGlyphPlacementDelta = new Point();
00909 
00910                         TrueTypeFile.PairKerning kerning =
    Font.FontFamily.TrueTypeFile.Get1000EmKerning(text[i], text[i + 1]);
00911 
00912                         if (kerning != null)
00913                         {
00914                             currentGlyphPlacementDelta = new Point(currentGlyphPlacementDelta.X +
    kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y + kerning.Glyph1Placement.Y);
00915                             currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
    kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
00916 
00917                             nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
    kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
00918                             nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
    kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
00919                         }
00920                     }
00921 
00922                     if (currentGlyphPlacementDelta.X != 0 || currentGlyphPlacementDelta.Y != 0 ||
    currentGlyphAdvanceDelta.X != 0 || currentGlyphAdvanceDelta.Y != 0)
00923                     {
00924                         if (currentRun.Length > 0)
00925                         {
00926                             tSpans.Add((currentRun.ToString(), currentKerning));
00927 
00928                             tSpans.Add((text[i].ToString(), new Point(currentGlyphPlacementDelta.X *
    Font.FontSize / 1000, currentGlyphPlacementDelta.Y * Font.FontSize / 1000)));
00929 
00930                             currentRun.Clear();
00931                             currentKerning = new Point((currentGlyphAdvanceDelta.X -
    currentGlyphPlacementDelta.X) * Font.FontSize / 1000, (currentGlyphAdvanceDelta.Y -
    currentGlyphPlacementDelta.Y) * Font.FontSize / 1000);
00932                         }
00933                         else
00934                         {
00935                             tSpans.Add((text[i].ToString(), new Point(currentGlyphPlacementDelta.X *
    Font.FontSize / 1000 + currentKerning.X, currentGlyphPlacementDelta.Y * Font.FontSize / 1000 +
    currentKerning.Y)));
00936 
00937                             currentRun.Clear();
00938                             currentKerning = new Point((currentGlyphAdvanceDelta.X -
    currentGlyphPlacementDelta.X) * Font.FontSize / 1000, (currentGlyphAdvanceDelta.Y -
    currentGlyphPlacementDelta.Y) * Font.FontSize / 1000);
00939                         }
```

```
00940                     }
00941                     else
00942                     {
00943                         currentRun.Append(text[i]);
00944                     }
00945                 }
00946 
00947                 if (currentRun.Length > 0)
00948                 {
00949                     tSpans.Add((currentRun.ToString(), currentKerning));
00950                 }
00951 
00952                 double currX = x;
00953                 double currY = y;
00954 
00955                 Font.DetailedFontMetrics fullMetrics = Font.MeasureTextAdvanced(text);
00956 
00957                 for (int i = 0; i < tSpans.Count; i++)
00958                 {
00959                     Font.DetailedFontMetrics metrics = Font.MeasureTextAdvanced(tSpans[i].Item1);
00960 
00961                     if (i == 0)
00962                     {
00963                         StrokeSimpleText(tSpans[i].Item1, currX + tSpans[i].Item2.X, currY +
      tSpans[i].Item2.Y);
00964                     }
00965                     else
00966                     {
00967                         StrokeSimpleText(tSpans[i].Item1, currX + metrics.LeftSideBearing -
      fullMetrics.LeftSideBearing + tSpans[i].Item2.X, currY + tSpans[i].Item2.Y);
00968                     }
00969 
00970 
00971                     currX += metrics.AdvanceWidth + tSpans[i].Item2.X;
00972                     currY += tSpans[i].Item2.Y;
00973                 }
00974             }
00975         }
00976 
00977 
00978         public void FillSimpleText(string text, double x, double y)
00979         {
00980             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00981 
00982             if ((_textOption == AvaloniaContextInterpreter.TextOptions.NeverConvert || (_textOption ==
      AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary && (Font.FontFamily.IsStandardFamily ||
      Font.FontFamily.TrueTypeFile?.FontStream != null))) &&
      !System.Runtime.InteropServices.RuntimeInformation.IsOSPlatform(System.Runtime.InteropServices.OSPlatform.Linux))
00983             {
00984                 SKTypeface typeface = SKTypefaceCache.GetSKTypeface(Font.FontFamily);
00985 
00986                 double top = y;
00987                 double left = x;
00988 
00989                 Font.DetailedFontMetrics metrics = Font.MeasureTextAdvanced(text);
00990 
00991                 if (TextBaseline == TextBaselines.Top)
00992                 {
00993                     if (Font.FontFamily.TrueTypeFile != null)
00994                     {
00995                         left -= metrics.LeftSideBearing;
00996                         top += metrics.Top;
00997                     }
00998                 }
00999                 else if (TextBaseline == TextBaselines.Middle)
01000                 {
01001                     if (Font.FontFamily.TrueTypeFile != null)
01002                     {
01003                         left -= metrics.LeftSideBearing;
01004                         top += (metrics.Top + metrics.Bottom) * 0.5;
01005                     }
01006                 }
01007                 else if (TextBaseline == TextBaselines.Baseline)
01008                 {
01009                     if (Font.FontFamily.TrueTypeFile != null)
01010                     {
01011                         left -= metrics.LeftSideBearing;
01012                     }
01013                 }
01014                 else if (TextBaseline == TextBaselines.Bottom)
01015                 {
01016                     if (Font.FontFamily.TrueTypeFile != null)
01017                     {
01018                         left -= metrics.LeftSideBearing;
01019                         top += metrics.Bottom;
01020                     }
01021                 }
```

```
01022
01023                 SKPaint fill = new SKPaint() { IsStroke = false, IsAntialias = true, Style =
      SKPaintStyle.Fill, SubpixelText = true };
01024
01025                 if (this.FillStyle is SolidColourBrush solid)
01026                 {
01027                     fill.Color = new SKColor((byte)(solid.R * 255), (byte)(solid.G * 255),
      (byte)(solid.B * 255), FillAlpha);
01028                 }
01029                 else if (this.FillStyle is LinearGradientBrush linearGradient)
01030                 {
01031                     fill.Shader = linearGradient.ToSKShader();
01032                 }
01033                 else if (this.FillStyle is RadialGradientBrush radialGradient)
01034                 {
01035                     fill.Shader = radialGradient.ToSKShader();
01036                 }
01037
01038                 SKRenderAction act = SKRenderAction.TextAction(text, (float)left, (float)top, new
      SKFont(typeface, (float)Font.FontSize), fill, Tag);
01039
01040                 AddAction(act);
01041             }
01042             else
01043             {
01044                 PathText(text, x, y);
01045                 Fill(FillRule.NonZeroWinding);
01046             }
01047         }
01048
01049         public Brush StrokeStyle { get; private set; } = Colour.FromRgb(0, 0, 0);
01050         private byte StrokeAlpha = 255;
01051
01052         public Brush FillStyle { get; private set; } = Colour.FromRgb(0, 0, 0);
01053         private byte FillAlpha = 255;
01054
01055         public void SetFillStyle((int r, int g, int b, double a) style)
01056         {
01057             FillStyle = Colour.FromRgba(style.r, style.g, style.b, (int)(style.a * 255));
01058             FillAlpha = (byte)(style.a * 255);
01059         }
01060
01061         public void SetFillStyle(Brush style)
01062         {
01063             FillStyle = style;
01064
01065             if (style is SolidColourBrush solid)
01066             {
01067                 FillAlpha = (byte)(solid.A * 255);
01068             }
01069             else
01070             {
01071                 FillAlpha = 255;
01072             }
01073         }
01074
01075         public void SetStrokeStyle((int r, int g, int b, double a) style)
01076         {
01077             StrokeStyle = Colour.FromRgba(style.r, style.g, style.b, (int)(style.a * 255));
01078             StrokeAlpha = (byte)(style.a * 255);
01079         }
01080
01081         public void SetStrokeStyle(Brush style)
01082         {
01083             StrokeStyle = style;
01084
01085             if (style is SolidColourBrush solid)
01086             {
01087                 StrokeAlpha = (byte)(solid.A * 255);
01088             }
01089             else
01090             {
01091                 StrokeAlpha = 255;
01092             }
01093         }
01094
01095         private LineDash LineDash;
01096
01097         public void SetLineDash(LineDash dash)
01098         {
01099             LineDash = new LineDash(dash.DashArray, dash.Phase);
01100         }
01101
01102         public void Rotate(double angle)
01103         {
01104             Utils.CoerceNaNAndInfinityToZero(ref angle);
01105
```

```
01106             SKRenderAction act = SKRenderAction.TransformAction(SKMatrix.CreateRotation((float)angle),
      Tag);
01107             AddAction(act);
01108 
01109             currentPath = null;
01110             figureInitialised = false;
01111         }
01112 
01113         public void Transform(double a, double b, double c, double d, double e, double f)
01114         {
01115             Utils.CoerceNaNAndInfinityToZero(ref a, ref b, ref c, ref d, ref e, ref f);
01116 
01117             SKRenderAction act = SKRenderAction.TransformAction(new SKMatrix((float)a, (float)c,
      (float)e, (float)b, (float)d, (float)f, 0, 0, 1), Tag);
01118             AddAction(act);
01119 
01120             currentPath = null;
01121             figureInitialised = false;
01122         }
01123 
01124         public void Scale(double x, double y)
01125         {
01126             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
01127 
01128             SKRenderAction act = SKRenderAction.TransformAction(SKMatrix.CreateScale((float)x,
      (float)y), Tag);
01129             AddAction(act);
01130 
01131             currentPath = null;
01132             figureInitialised = false;
01133         }
01134 
01135         public void Save()
01136         {
01137             SKRenderAction act = SKRenderAction.SaveAction(Tag);
01138             AddAction(act);
01139 
01140             currentPath = null;
01141             figureInitialised = false;
01142         }
01143 
01144         public void Restore()
01145         {
01146             SKRenderAction act = SKRenderAction.RestoreAction(Tag);
01147             AddAction(act);
01148 
01149             currentPath = null;
01150             figureInitialised = false;
01151         }
01152 
01153         public double LineWidth { get; set; }
01154         public LineCaps LineCap { get; set; }
01155         public LineJoins LineJoin { get; set; }
01156 
01157         public Font Font { get; set; }
01158 
01159         private SKPath currentPath;
01160 
01161         private bool figureInitialised = false;
01162         private bool disposedValue;
01163 
01164         public void MoveTo(double x, double y)
01165         {
01166             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
01167 
01168             if (currentPath == null)
01169             {
01170                 currentPath = new SKPath() { FillType = SKPathFillType.EvenOdd };
01171             }
01172 
01173             currentPath.MoveTo((float)x, (float)y);
01174             figureInitialised = true;
01175         }
01176 
01177         public void LineTo(double x, double y)
01178         {
01179             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
01180 
01181             if (currentPath == null)
01182             {
01183                 currentPath = new SKPath() { FillType = SKPathFillType.EvenOdd };
01184             }
01185 
01186             if (!figureInitialised)
01187             {
01188                 figureInitialised = true;
01189                 currentPath.MoveTo((float)x, (float)y);
```

```
01190             }
01191             else
01192             {
01193                 currentPath.LineTo((float)x, (float)y);
01194             }
01195         }
01196 
01197         public void Rectangle(double x0, double y0, double width, double height)
01198         {
01199             Utils.CoerceNaNAndInfinityToZero(ref x0, ref y0, ref width, ref height);
01200 
01201             if (currentPath == null)
01202             {
01203                 currentPath = new SKPath() { FillType = SKPathFillType.EvenOdd };
01204             }
01205 
01206             currentPath.MoveTo((float)x0, (float)y0);
01207             currentPath.LineTo((float)(x0 + width), (float)y0);
01208             currentPath.LineTo((float)(x0 + width), (float)(y0 + height));
01209             currentPath.LineTo((float)x0, (float)(y0 + height));
01210 
01211             currentPath.Close();
01212             figureInitialised = false;
01213         }
01214 
01215         public void CubicBezierTo(double p1X, double p1Y, double p2X, double p2Y, double p3X, double
      p3Y)
01216         {
01217             Utils.CoerceNaNAndInfinityToZero(ref p1X, ref p1Y, ref p2X, ref p2Y, ref p3X, ref p3Y);
01218 
01219             if (currentPath == null)
01220             {
01221                 currentPath = new SKPath() { FillType = SKPathFillType.EvenOdd };
01222             }
01223 
01224             if (figureInitialised)
01225             {
01226                 currentPath.CubicTo((float)p1X, (float)p1Y, (float)p2X, (float)p2Y, (float)p3X,
      (float)p3Y);
01227             }
01228             else
01229             {
01230                 currentPath.MoveTo((float)p1X, (float)p1Y);
01231                 figureInitialised = true;
01232             }
01233         }
01234 
01235         public void Close()
01236         {
01237             currentPath.Close();
01238 
01239             figureInitialised = false;
01240         }
01241 
01242         public void Stroke()
01243         {
01244             SKPaint stroke = new SKPaint() { IsStroke = true, IsAntialias = true, Style =
      SKPaintStyle.Stroke, StrokeWidth = (float)LineWidth };
01245 
01246             if (this.StrokeStyle is SolidColourBrush solid)
01247             {
01248                 stroke.Color = new SKColor((byte)(solid.R * 255), (byte)(solid.G * 255),
      (byte)(solid.B * 255), StrokeAlpha);
01249             }
01250             else if (this.StrokeStyle is LinearGradientBrush linearGradient)
01251             {
01252                 stroke.Shader = linearGradient.ToSKShader();
01253             }
01254             else if (this.StrokeStyle is RadialGradientBrush radialGradient)
01255             {
01256                 stroke.Shader = radialGradient.ToSKShader();
01257             }
01258 
01259             stroke.PathEffect = SKPathEffect.CreateDash(LineDash.DashArray.Select(x =>
      (float)x).ToArray(), (float)LineDash.Phase);
01260 
01261             switch (LineCap)
01262             {
01263                 case LineCaps.Butt:
01264                     stroke.StrokeCap = SKStrokeCap.Butt;
01265                     break;
01266                 case LineCaps.Round:
01267                     stroke.StrokeCap = SKStrokeCap.Round;
01268                     break;
01269                 case LineCaps.Square:
01270                     stroke.StrokeCap = SKStrokeCap.Square;
01271                     break;
```

```
01272             }
01273
01274             switch (LineJoin)
01275             {
01276                 case LineJoins.Bevel:
01277                     stroke.StrokeJoin = SKStrokeJoin.Bevel;
01278                     break;
01279                 case LineJoins.Round:
01280                     stroke.StrokeJoin = SKStrokeJoin.Round;
01281                     break;
01282                 case LineJoins.Miter:
01283                     stroke.StrokeJoin = SKStrokeJoin.Miter;
01284                     break;
01285             }
01286
01287             SKRenderAction act = SKRenderAction.PathAction(currentPath, stroke, Tag);
01288
01289             AddAction(act);
01290
01291             currentPath = null;
01292             figureInitialised = false;
01293         }
01294
01295         public void Fill(FillRule fillRule)
01296         {
01297             SKPaint fill = new SKPaint() { IsStroke = false, IsAntialias = true, Style =
      SKPaintStyle.Fill };
01298
01299             if (this.FillStyle is SolidColourBrush solid)
01300             {
01301                 fill.Color = new SKColor((byte)(solid.R * 255), (byte)(solid.G * 255), (byte)(solid.B
      * 255), FillAlpha);
01302             }
01303             else if (this.FillStyle is LinearGradientBrush linearGradient)
01304             {
01305                 fill.Shader = linearGradient.ToSKShader();
01306             }
01307             else if (this.FillStyle is RadialGradientBrush radialGradient)
01308             {
01309                 fill.Shader = radialGradient.ToSKShader();
01310             }
01311
01312             switch (fillRule)
01313             {
01314                 case FillRule.NonZeroWinding:
01315                     currentPath.FillType = SKPathFillType.Winding;
01316                     break;
01317
01318                 case FillRule.EvenOdd:
01319                     currentPath.FillType = SKPathFillType.EvenOdd;
01320                     break;
01321             }
01322
01323             SKRenderAction act = SKRenderAction.PathAction(currentPath, fill, Tag);
01324
01325             AddAction(act);
01326
01327             currentPath = null;
01328             figureInitialised = false;
01329         }
01330
01331         public void SetClippingPath()
01332         {
01333             SKRenderAction act = SKRenderAction.ClipAction(currentPath, Tag);
01334
01335             AddAction(act);
01336
01337             currentPath = null;
01338             figureInitialised = false;
01339         }
01340
01341         public void DrawRasterImage(int sourceX, int sourceY, int sourceWidth, int sourceHeight,
      double destinationX, double destinationY, double destinationWidth, double destinationHeight,
      RasterImage image)
01342         {
01343             Utils.CoerceNaNAndInfinityToZero(ref destinationX, ref destinationY, ref destinationWidth,
      ref destinationHeight);
01344
01345             if (!this.Images.ContainsKey(image.Id))
01346             {
01347                 SKBitmap bmp = SKBitmap.Decode(image.PNGStream);
01348                 this.Images.Add(image.Id, (bmp, image.Interpolate));
01349             }
01350
01351             SKRenderAction act = SKRenderAction.ImageAction(image.Id, new SKRect(sourceX, sourceY,
      sourceX + sourceWidth, sourceY + sourceHeight), new SKRect((float)destinationX, (float)destinationY,
      (float)(destinationX + destinationWidth), (float)(destinationY + destinationHeight)), Tag);
```

```
01352
01353             AddAction(act);
01354         }
01355
01356         public void DrawFilteredGraphics(Graphics graphics, IFilter filter)
01357         {
01358             if (this._filterOption.Operation == FilterOption.FilterOperations.RasteriseAllWithSkia)
01359             {
01360                 double scale = this._filterOption.RasterisationResolution;
01361
01362                 Rectangle bounds = graphics.GetBounds();
01363
01364                 bounds = new Rectangle(bounds.Location.X - filter.TopLeftMargin.X, bounds.Location.Y -
      filter.TopLeftMargin.Y, bounds.Size.Width + filter.TopLeftMargin.X + filter.BottomRightMargin.X,
      bounds.Size.Height + filter.TopLeftMargin.Y + filter.BottomRightMargin.Y);
01365
01366                 if (bounds.Size.Width > 0 && bounds.Size.Height > 0)
01367                 {
01368                     if (!this._filterOption.RasterisationResolutionRelative)
01369                     {
01370                         scale = scale / Math.Min(bounds.Size.Width, bounds.Size.Height);
01371                     }
01372
01373                     RasterImage rasterised = SKRenderContextInterpreter.Rasterise(graphics, bounds,
      scale, true);
01374                     RasterImage filtered = null;
01375
01376                     if (filter is IFilterWithRasterisableParameter filterWithRastParam)
01377                     {
01378                         filterWithRastParam.RasteriseParameter(SKRenderContextInterpreter.Rasterise,
      scale);
01379                     }
01380
01381                     if (filter is ILocationInvariantFilter locInvFilter)
01382                     {
01383                         filtered = locInvFilter.Filter(rasterised, scale);
01384                     }
01385                     else if (filter is IFilterWithLocation filterWithLoc)
01386                     {
01387                         filtered = filterWithLoc.Filter(rasterised, bounds, scale);
01388                     }
01389
01390                     if (filtered != null)
01391                     {
01392                         rasterised.Dispose();
01393
01394                         DrawRasterImage(0, 0, filtered.Width, filtered.Height, bounds.Location.X,
      bounds.Location.Y, bounds.Size.Width, bounds.Size.Height, filtered);
01395                     }
01396                 }
01397             }
01398             else if (this._filterOption.Operation ==
      FilterOption.FilterOperations.RasteriseAllWithVectSharp)
01399             {
01400                 double scale = this._filterOption.RasterisationResolution;
01401
01402                 Rectangle bounds = graphics.GetBounds();
01403
01404                 bounds = new Rectangle(bounds.Location.X - filter.TopLeftMargin.X, bounds.Location.Y -
      filter.TopLeftMargin.Y, bounds.Size.Width + filter.TopLeftMargin.X + filter.BottomRightMargin.X,
      bounds.Size.Height + filter.TopLeftMargin.Y + filter.BottomRightMargin.Y);
01405
01406                 if (bounds.Size.Width > 0 && bounds.Size.Height > 0)
01407                 {
01408                     if (!this._filterOption.RasterisationResolutionRelative)
01409                     {
01410                         scale = scale / Math.Min(bounds.Size.Width, bounds.Size.Height);
01411                     }
01412
01413                     if (graphics.TryRasterise(bounds, scale, true, out RasterImage rasterised))
01414                     {
01415                         RasterImage filtered = null;
01416
01417                         if (filter is IFilterWithRasterisableParameter filterWithRastParam)
01418                         {
01419
      filterWithRastParam.RasteriseParameter(SKRenderContextInterpreter.Rasterise, scale);
01420                         }
01421
01422                         if (filter is ILocationInvariantFilter locInvFilter)
01423                         {
01424                             filtered = locInvFilter.Filter(rasterised, scale);
01425                         }
01426                         else if (filter is IFilterWithLocation filterWithLoc)
01427                         {
01428                             filtered = filterWithLoc.Filter(rasterised, bounds, scale);
01429                         }
```

```
01430
01431                         if (filtered != null)
01432                         {
01433                             rasterised.Dispose();
01434
01435                             DrawRasterImage(0, 0, filtered.Width, filtered.Height, bounds.Location.X,
      bounds.Location.Y, bounds.Size.Width, bounds.Size.Height, filtered);
01436                         }
01437                     }
01438                     else
01439                     {
01440                         throw new NotImplementedException(@"The filter could not be rasterised!  You
      can avoid this error by doing one of the following:
01441 • Add a reference to VectSharp.Raster or VectSharp.Raster.ImageSharp (you may also need to add a using
      directive somewhere to force the assembly to be loaded).
01442 • Provide your own implementation of Graphics.RasterisationMethod.
01443 • Set the FilterOption.Operation to ""RasteriseAllWithSkia"", ""IgnoreAll"" or ""SkipAll"".");
01444                     }
01445                 }
01446             }
01447             else if (this._filterOption.Operation == FilterOption.FilterOperations.IgnoreAll)
01448             {
01449                 graphics.CopyToIGraphicsContext(this);
01450             }
01451             else
01452             {
01453
01454             }
01455         }
01456
01457         protected virtual void Dispose(bool disposing)
01458         {
01459             if (!disposedValue)
01460             {
01461                 if (disposing)
01462                 {
01463                     this.currentPath?.Dispose();
01464                 }
01465
01466                 disposedValue = true;
01467             }
01468         }
01469
01470         public void Dispose()
01471         {
01472             Dispose(disposing:  true);
01473             GC.SuppressFinalize(this);
01474         }
01475     }
01476
01477 /// <summary>
01478 /// Determines how and whether image filters are rasterised.
01479 /// </summary>
01480     public class FilterOption
01481     {
01482 /// <summary>
01483 /// Defines whether image filters should be rasterised or not.
01484 /// </summary>
01485         public enum FilterOperations
01486         {
01487 /// <summary>
01488 /// Image filters will always be rasterised using the SkiaSharp backend.
01489 /// </summary>
01490             RasteriseAllWithSkia,
01491
01492 /// <summary>
01493 /// Image filters will always be rasterised using the VectSharp.Raster or VectSharp.Raster.ImageSharp.
      This option requires a reference to VectSharp.Raster or to VectSharp.Raster.ImageSharp to be added.
01494 /// </summary>
01495             RasteriseAllWithVectSharp,
01496
01497 /// <summary>
01498 /// All image filters will be ignored.
01499 /// </summary>
01500             IgnoreAll,
01501
01502 /// <summary>
01503 /// All the images that should be drawn with a filter will be ignored.
01504 /// </summary>
01505             SkipAll
01506         }
01507
01508 /// <summary>
01509 /// Defines whether image filters should be rasterised or not.
01510 /// </summary>
01511         public FilterOperations Operation { get; } = FilterOperations.RasteriseAllWithSkia;
01512
```

```
01513 /// <summary>
01514 /// The resolution that will be used to rasterise image filters.  Depending on the value of <see
      cref="RasterisationResolutionRelative"/>, this can either be an absolute resolution (i.e.  a size in
      pixel), or a scale factor that is applied to the image size in graphics units.
01515 /// </summary>
01516         public double RasterisationResolution { get; } = 1;
01517
01518 /// <summary>
01519 /// Determines whether the value of <see cref="RasterisationResolution"/> is absolute (i.e.  a size in
      pixel), or relative (i.e.  a scale factor that is applied to the image size in graphics units).
01520 /// </summary>
01521         public bool RasterisationResolutionRelative { get; } = true;
01522
01523 /// <summary>
01524 /// The default options for image filter rasterisation.
01525 /// </summary>
01526         public static FilterOption Default = new FilterOption(FilterOperations.RasteriseAllWithSkia,
      1, true);
01527
01528 /// <summary>
01529 /// Create a new <see cref="FilterOption"/> object.
01530 /// </summary>
01531 /// <param name="operation">Defines whether image filters should be rasterised or not.</param>
01532 /// <param name="rasterisationResolution">The resolution that will be used to rasterise image filters.
      Depending on the value of <see cref="RasterisationResolutionRelative"/>, this can either be an
      absolute resolution (i.e.  a size in pixel), or a scale factor that is applied to the image size in
      graphics units.</param>
01533 /// <param name="rasterisationResolutionRelative">Determines whether the value of <see
      cref="RasterisationResolution"/> is absolute (i.e.  a size in pixel), or relative (i.e.  a scale
      factor that is applied to the image size in graphics units).</param>
01534         public FilterOption(FilterOperations operation, double rasterisationResolution, bool
      rasterisationResolutionRelative)
01535         {
01536             this.Operation = operation;
01537             this.RasterisationResolution = rasterisationResolution;
01538             this.RasterisationResolutionRelative = rasterisationResolutionRelative;
01539         }
01540     }
01541
01542 /// <summary>
01543 /// Contains methods to render a <see cref="Page"/> to an <see cref="Avalonia.Controls.Canvas"/> using
      the SkiaSharp renderer.
01544 /// </summary>
01545     public static class SKRenderContextInterpreter
01546     {
01547         internal static SKColor ToSKColor(this Colour colour)
01548         {
01549             return new SKColor((byte)(255 * colour.R), (byte)(255 * colour.G), (byte)(255 * colour.B),
      (byte)(255 * colour.A));
01550         }
01551
01552         internal static SKShader ToSKShader(this LinearGradientBrush brush)
01553         {
01554             return SKShader.CreateLinearGradient(new SKPoint((float)brush.StartPoint.X,
      (float)brush.StartPoint.Y), new SKPoint((float)brush.EndPoint.X, (float)brush.EndPoint.Y), (from el in
      brush.GradientStops select el.Colour.ToSKColor()).ToArray(), (from el in brush.GradientStops select
      (float)el.Offset).ToArray(), SKShaderTileMode.Clamp);
01555         }
01556
01557         internal static SKShader ToSKShader(this RadialGradientBrush brush)
01558         {
01559             return SKShader.CreateTwoPointConicalGradient(new SKPoint((float)brush.FocalPoint.X,
      (float)brush.FocalPoint.Y), 0, new SKPoint((float)brush.Centre.X, (float)brush.Centre.Y),
      (float)brush.Radius, (from el in brush.GradientStops select el.Colour.ToSKColor()).ToArray(), (from el
      in brush.GradientStops select (float)el.Offset).ToArray(), SKShaderTileMode.Clamp);
01560         }
01561
01562 /// <summary>
01563 /// Render a <see cref="Document"/> to an <see cref="Avalonia.Controls.Canvas"/> using the SkiaSharp
      renderer.  Each page corresponds to a layer in the image.
01564 /// </summary>
01565 /// <param name="document">The <see cref="Document"/> to render.</param>
01566 /// <param name="width">The width of the document.  If this is <see langword="null" />, the width of
      the largest page is used.</param>
01567 /// <param name="height">The height of the document.  If this is <see langword="null" />, the height
      of the largest page is used.</param>
01568 /// <param name="background">The background colour of the document.  If this is <see langword="null"
      />, a transparent background is used.</param>
01569 /// <param name="textOption">Defines whether text items should be converted into paths when
      drawing.</param>
01570 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
01571 /// <returns>An <see cref="Avalonia.Controls.Canvas"/> containing the rendered graphics
      objects.</returns>
01572         public static SKMultiLayerRenderCanvas PaintToSKCanvas(this Document document, double?  width
      = null, double?  height = null, Colour?  background = null, AvaloniaContextInterpreter.TextOptions
      textOption = AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption =
```

```
default)
01573         {
01574             filterOption = filterOption ??  FilterOption.Default;
01575 
01576             return new SKMultiLayerRenderCanvas((from el in document.Pages select
el.CopyToSKRenderContext(textOption, filterOption)).ToList(), (from el in document.Pages select
SKRenderAction.TransformAction(SKMatrix.Identity)).ToList(), background ??  Colour.FromRgba(0, 0, 0,
0), width ??  (from el in document.Pages select el.Width).Max(), height ??  (from el in document.Pages
select el.Height).Max());
01577         }
01578 
01579 /// <summary>
01580 /// Render a <see cref="Document"/> to an <see cref="Avalonia.Controls.Canvas"/> using the SkiaSharp
renderer.  Each page corresponds to a layer in the image.
01581 /// </summary>
01582 /// <param name="document">The <see cref="Document"/> to render.</param>
01583 /// <param name="taggedActions">A Dictionary containing the actions that will be performed on items
with the corresponding tag.
01584 /// These should be functions that accept one parameter of type <see cref="SKRenderAction"/> and
return an <see cref="IEnumerable{SKRenderAction}"/> of the render actions that will actually be added
to the plot.</param>
01585 /// <param name="removeTaggedActionsAfterExecution">Whether the actions should be removed from
<paramref name="taggedActions"/> after their execution.  Set to false if the same action should be
performed on multiple items with the same tag.</param>
01586 /// <param name="width">The width of the document.  If this is <see langword="null" />, the width of
the largest page is used.</param>
01587 /// <param name="height">The height of the document.  If this is <see langword="null" />, the height
of the largest page is used.</param>
01588 /// <param name="background">The background colour of the document.  If this is <see langword="null"
/>, a transparent background is used.</param>
01589 /// <param name="textOption">Defines whether text items should be converted into paths when
drawing.</param>
01590 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
rendering the image.</param>
01591 /// <returns>An <see cref="Avalonia.Controls.Canvas"/> containing the rendered graphics
objects.</returns>
01592         public static SKMultiLayerRenderCanvas PaintToSKCanvas(this Document document,
Dictionary<string, Func<SKRenderAction, IEnumerable<SKRenderAction>> taggedActions, bool
removeTaggedActionsAfterExecution = true, double?  width = null, double?  height = null, Colour?
background = null, AvaloniaContextInterpreter.TextOptions textOption =
AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption = default)
01593         {
01594             filterOption = filterOption ??  FilterOption.Default;
01595 
01596             return new SKMultiLayerRenderCanvas((from el in document.Pages select
el.CopyToSKRenderContext(taggedActions, removeTaggedActionsAfterExecution, textOption,
filterOption)).ToList(), (from el in document.Pages select
SKRenderAction.TransformAction(SKMatrix.Identity)).ToList(), background ??  Colour.FromRgba(0, 0, 0,
0), width ??  (from el in document.Pages select el.Width).Max(), height ??  (from el in document.Pages
select el.Height).Max());
01597         }
01598 
01599 /// <summary>
01600 /// Render a <see cref="Document"/> to an <see cref="Avalonia.Controls.Canvas"/> using the SkiaSharp
renderer.  Each page corresponds to a layer in the image.
01601 /// </summary>
01602 /// <param name="document">The <see cref="Document"/> to render.</param>
01603 /// <param name="taggedActions">A Dictionary containing the actions that will be performed on items
with the corresponding tag.
01604 /// These should be functions that accept one parameter of type <see cref="SKRenderAction"/> and
return an <see cref="IEnumerable{SKRenderAction}"/> of the render actions that will actually be added
to the plot.</param>
01605 /// <param name="images">A dictionary that associates to each raster image path (or data URL) the
image rendered as a <see cref="SKBitmap"/> and a boolean value indicating whether it should be drawn
as "pixelated" or not.  This will be populated automatically as the page is rendered.
01606 /// If you are rendering multiple <see cref="Page"/>s (or you are rendering the same page multiple
times), it will be beneficial to keep a reference to this dictionary and pass it again on further
rendering requests; otherwise, you can just pass an empty dictionary.</param>
01607 /// <param name="removeTaggedActionsAfterExecution">Whether the actions should be removed from
<paramref name="taggedActions"/> after their execution.  Set to false if the same action should be
performed on multiple items with the same tag.</param>
01608 /// <param name="width">The width of the document.  If this is <see langword="null" />, the width of
the largest page is used.</param>
01609 /// <param name="height">The height of the document.  If this is <see langword="null" />, the height
of the largest page is used.</param>
01610 /// <param name="background">The background colour of the document.  If this is <see langword="null"
/>, a transparent background is used.</param>
01611 /// <param name="textOption">Defines whether text items should be converted into paths when
drawing.</param>
01612 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
rendering the image.</param>
01613 /// <returns>An <see cref="Avalonia.Controls.Canvas"/> containing the rendered graphics
objects.</returns>
01614         public static SKMultiLayerRenderCanvas PaintToSKCanvas(this Document document,
Dictionary<string, Func<SKRenderAction, IEnumerable<SKRenderAction>> taggedActions, Dictionary<string,
(SKBitmap, bool)> images, bool removeTaggedActionsAfterExecution = true, double?  width = null,
double?  height = null, Colour?  background = null, AvaloniaContextInterpreter.TextOptions textOption
```

```
      = AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption = default)
01615         {
01616             filterOption = filterOption ??  FilterOption.Default;
01617 
01618             return new SKMultiLayerRenderCanvas((from el in document.Pages select
      el.CopyToSKRenderContext(taggedActions, images, removeTaggedActionsAfterExecution, textOption,
      filterOption)).ToList(), (from el in document.Pages select
      SKRenderAction.TransformAction(SKMatrix.Identity)).ToList(), background ??  Colour.FromRgba(0, 0, 0,
      0), width ??  (from el in document.Pages select el.Width).Max(), height ??  (from el in document.Pages
      select el.Height).Max());
01619         }
01620 
01621 /// <summary>
01622 /// Render a <see cref="Page"/> to an <see cref="Avalonia.Controls.Canvas"/> using the SkiaSharp
      renderer.
01623 /// </summary>
01624 /// <param name="page">The <see cref="Page"/> to render.</param>
01625 /// <param name="textOption">Defines whether text items should be converted into paths when
      drawing.</param>
01626 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
01627 /// <returns>An <see cref="Avalonia.Controls.Canvas"/> containing the rendered graphics
      objects.</returns>
01628         public static SKMultiLayerRenderCanvas PaintToSKCanvas(this Page page,
      AvaloniaContextInterpreter.TextOptions textOption =
      AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption = default)
01629         {
01630             filterOption = filterOption ??  FilterOption.Default;
01631 
01632             return new SKMultiLayerRenderCanvas(new List<SKRenderContext>() {
      page.CopyToSKRenderContext(textOption, filterOption) }, new List<SKRenderAction>() {
      SKRenderAction.TransformAction(SKMatrix.Identity) }, page.Background, page.Width, page.Height);
01633         }
01634 
01635 /// <summary>
01636 /// Render a <see cref="Page"/> to an <see cref="Avalonia.Controls.Canvas"/> using the
      SkiaSharpRenderer.
01637 /// </summary>
01638 /// <param name="page">The <see cref="Page"/> to render.</param>
01639 /// <param name="taggedActions">A Dictionary containing the actions that will be performed on items
      with the corresponding tag.
01640 /// These should be functions that accept one parameter of type <see cref="SKRenderAction"/> and
      return an <see cref="IEnumerable{SKRenderAction}"/> of the render actions that will actually be added
      to the plot.</param>
01641 /// <param name="removeTaggedActionsAfterExecution">Whether the actions should be removed from
      <paramref name="taggedActions"/> after their execution.  Set to false if the same action should be
      performed on multiple items with the same tag.</param>
01642 /// <param name="textOption">Defines whether text items should be converted into paths when
      drawing.</param>
01643 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
01644 /// <returns>An <see cref="Avalonia.Controls.Canvas"/> containing the rendered graphics
      objects.</returns>
01645         public static SKMultiLayerRenderCanvas PaintToSKCanvas(this Page page, Dictionary<string,
      Func<SKRenderAction, IEnumerable<SKRenderAction>> taggedActions, bool
      removeTaggedActionsAfterExecution = true, AvaloniaContextInterpreter.TextOptions textOption =
      AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption = default)
01646         {
01647             filterOption = filterOption ??  FilterOption.Default;
01648 
01649             return new SKMultiLayerRenderCanvas(new List<SKRenderContext>() {
      page.CopyToSKRenderContext(taggedActions, removeTaggedActionsAfterExecution, textOption, filterOption)
      }, new List<SKRenderAction>() { SKRenderAction.TransformAction(SKMatrix.Identity) }, page.Background,
      page.Width, page.Height);
01650         }
01651 
01652 /// <summary>
01653 /// Render a <see cref="Page"/> to an <see cref="Avalonia.Controls.Canvas"/> using the
      SkiaSharpRenderer.
01654 /// </summary>
01655 /// <param name="page">The <see cref="Page"/> to render.</param>
01656 /// <param name="taggedActions">A Dictionary containing the actions that will be performed on items
      with the corresponding tag.
01657 /// These should be functions that accept one parameter of type <see cref="SKRenderAction"/> and
      return an <see cref="IEnumerable{SKRenderAction}"/> of the render actions that will actually be added
      to the plot.</param>
01658 /// <param name="images">A dictionary that associates to each raster image path (or data URL) the
      image rendered as a <see cref="SKBitmap"/> and a boolean value indicating whether it should be drawn
      as "pixelated" or not.  This will be populated automatically as the page is rendered.
01659 /// If you are rendering multiple <see cref="Page"/>s (or you are rendering the same page multiple
      times), it will be beneficial to keep a reference to this dictionary and pass it again on further
      rendering requests; otherwise, you can just pass an empty dictionary.</param>
01660 /// <param name="removeTaggedActionsAfterExecution">Whether the actions should be removed from
      <paramref name="taggedActions"/> after their execution.  Set to false if the same action should be
      performed on multiple items with the same tag.</param>
01661 /// <param name="textOption">Defines whether text items should be converted into paths when
      drawing.</param>
```

```
01662 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
01663 /// <returns>An <see cref="Avalonia.Controls.Canvas"/> containing the rendered graphics
      objects.</returns>
01664         public static SKMultiLayerRenderCanvas PaintToSKCanvas(this Page page, Dictionary<string,
      Func<SKRenderAction, IEnumerable<SKRenderAction>> taggedActions, Dictionary<string, (SKBitmap, bool)>
      images, bool removeTaggedActionsAfterExecution = true, AvaloniaContextInterpreter.TextOptions
      textOption = AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption =
      default)
01665         {
01666             filterOption = filterOption ??  FilterOption.Default;
01667 
01668             return new SKMultiLayerRenderCanvas(new List<SKRenderContext>() {
      page.CopyToSKRenderContext(taggedActions, images, removeTaggedActionsAfterExecution, textOption,
      filterOption) }, new List<SKRenderAction>() { SKRenderAction.TransformAction(SKMatrix.Identity) },
      page.Background, page.Width, page.Height);
01669         }
01670 
01671 /// <summary>
01672 /// Render a <see cref="Page"/> to a <see cref="SKRenderContext"/>.  This can be drawn using the
      SkiaSharpRenderer by adding it to a <see cref="SKMultiLayerRenderCanvas"/>.
01673 /// </summary>
01674 /// <param name="page">The <see cref="Page"/> to render.</param>
01675 /// <param name="textOption">Defines whether text items should be converted into paths when
      drawing.</param>
01676 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
01677 /// <returns>A <see cref="SKRenderContext"/> containing the rendered graphics objects.</returns>
01678         public static SKRenderContext CopyToSKRenderContext(this Page page,
      AvaloniaContextInterpreter.TextOptions textOption =
      AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption = default)
01679         {
01680             filterOption = filterOption ??  FilterOption.Default;
01681 
01682             return CopyToSKRenderContext(page, new Dictionary<string, Func<SKRenderAction,
      IEnumerable<SKRenderAction>>(), new Dictionary<string, (SKBitmap, bool)>(), textOption:  textOption,
      filterOption:  filterOption);
01683         }
01684 
01685 /// <summary>
01686 /// Render a <see cref="Page"/> to a <see cref="SKRenderContext"/>.  This can be drawn using the
      SkiaSharpRenderer by adding it to a <see cref="SKMultiLayerRenderCanvas"/>.
01687 /// </summary>
01688 /// <param name="page">The <see cref="Page"/> to render.</param>
01689 /// <param name="taggedActions">A Dictionary containing the actions that will be performed on items
      with the corresponding tag.
01690 /// These should be functions that accept one parameter of type <see cref="SKRenderAction"/> and
      return an <see cref="IEnumerable{SKRenderAction}"/> of the render actions that will actually be added
      to the plot.</param>
01691 /// <param name="removeTaggedActionsAfterExecution">Whether the actions should be removed from
      <paramref name="taggedActions"/> after their execution.  Set to false if the same action should be
      performed on multiple items with the same tag.</param>
01692 /// <param name="textOption">Defines whether text items should be converted into paths when
      drawing.</param>
01693 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
01694 /// <returns>A <see cref="SKRenderContext"/> containing the rendered graphics objects.</returns>
01695         public static SKRenderContext CopyToSKRenderContext(this Page page, Dictionary<string,
      Func<SKRenderAction, IEnumerable<SKRenderAction>> taggedActions, bool
      removeTaggedActionsAfterExecution = true, AvaloniaContextInterpreter.TextOptions textOption =
      AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption = default)
01696         {
01697             filterOption = filterOption ??  FilterOption.Default;
01698 
01699             return CopyToSKRenderContext(page, taggedActions, new Dictionary<string, (SKBitmap,
      bool)>(), removeTaggedActionsAfterExecution, textOption, filterOption);
01700         }
01701 
01702 /// <summary>
01703 /// Render a <see cref="Page"/> to a <see cref="SKRenderContext"/>.  This can be drawn using the
      SkiaSharpRenderer by adding it to a <see cref="SKMultiLayerRenderCanvas"/>.
01704 /// </summary>
01705 /// <param name="page">The <see cref="Page"/> to render.</param>
01706 /// <param name="taggedActions">A Dictionary containing the actions that will be performed on items
      with the corresponding tag.
01707 /// These should be functions that accept one parameter of type <see cref="SKRenderAction"/> and
      return an <see cref="IEnumerable{SKRenderAction}"/> of the render actions that will actually be added
      to the plot.</param>
01708 /// <param name="images">A dictionary that associates to each raster image path (or data URL) the
      image rendered as a <see cref="SKBitmap"/> and a boolean value indicating whether it should be drawn
      as "pixelated" or not.  This will be populated automatically as the page is rendered.
01709 /// If you are rendering multiple <see cref="Page"/>s (or you are rendering the same page multiple
      times), it will be beneficial to keep a reference to this dictionary and pass it again on further
      rendering requests; otherwise, you can just pass an empty dictionary.</param>
01710 /// <param name="removeTaggedActionsAfterExecution">Whether the actions should be removed from
      <paramref name="taggedActions"/> after their execution.  Set to false if the same action should be
      performed on multiple items with the same tag.</param>
```

```
01711 /// <param name="textOption">Defines whether text items should be converted into paths when
      drawing.</param>
01712 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
01713 /// <returns>A <see cref="SKRenderContext"/> containing the rendered graphics objects.</returns>
01714         public static SKRenderContext CopyToSKRenderContext(this Page page, Dictionary<string,
      Func<SKRenderAction, IEnumerable<SKRenderAction>> taggedActions, Dictionary<string, (SKBitmap, bool)>
      images, bool removeTaggedActionsAfterExecution = true, AvaloniaContextInterpreter.TextOptions
      textOption = AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary, FilterOption filterOption =
      default)
01715         {
01716             filterOption = filterOption ??  FilterOption.Default;
01717 
01718             SKRenderContextImpl tbr = new SKRenderContextImpl(page.Width, page.Height,
      removeTaggedActionsAfterExecution, textOption, images, filterOption)
01719             {
01720                 TaggedActions = taggedActions
01721             };
01722             page.Graphics.CopyToIGraphicsContext(tbr);
01723 
01724             return tbr;
01725         }
01726 
01727 
01728 /// <summary>
01729 /// Rasterise a region of a <see cref="Graphics"/> object.
01730 /// </summary>
01731 /// <param name="graphics">The <see cref="Graphics"/> object that will be rasterised.</param>
01732 /// <param name="region">The region of the <paramref name="graphics"/> that will be
      rasterised.</param>
01733 /// <param name="scale">The scale at which the image will be rendered.</param>
01734 /// <param name="interpolate">Whether the resulting image should be interpolated or not when it is
      drawn on another <see cref="Graphics"/> surface.</param>
01735 /// <returns>A <see cref="RasterImage"/> containing the rasterised graphics.</returns>
01736         public static RasterImage Rasterise(this Graphics graphics, Rectangle region, double scale,
      bool interpolate)
01737         {
01738             Page pag = new Page(1, 1);
01739             pag.Graphics.DrawGraphics(0, 0, graphics);
01740             pag.Crop(region.Location, region.Size);
01741 
01742             //pag.PaintToSKCanvas().RenderAtResolution((int)Math.Round(region.Size.Width * scale),
      (int)Math.Round(region.Size.Height * scale), null);
01743 
01744             int width = (int)Math.Round(region.Size.Width * scale);
01745             int height = (int)Math.Round(region.Size.Height * scale);
01746 
01747             SKRenderContext ctx = pag.CopyToSKRenderContext();
01748 
01749             SKBitmap bitmap = new SKBitmap(width, height, SKColorType.Rgba8888, SKAlphaType.Unpremul);
01750 
01751             SKCanvas canvas = new SKCanvas(bitmap);
01752 
01753             canvas.Save();
01754 
01755             for (int i = 0; i < ctx.SKRenderActions.Count; i++)
01756             {
01757                 if (ctx.SKRenderActions[i].ActionType == SKRenderAction.ActionTypes.Clip)
01758                 {
01759                     canvas.ClipPath(ctx.SKRenderActions[i].Path, antialias:  true);
01760                 }
01761                 else if (ctx.SKRenderActions[i].ActionType == SKRenderAction.ActionTypes.Restore)
01762                 {
01763                     canvas.Restore();
01764                 }
01765                 else if (ctx.SKRenderActions[i].ActionType == SKRenderAction.ActionTypes.Save)
01766                 {
01767                     canvas.Save();
01768                 }
01769                 else if (ctx.SKRenderActions[i].ActionType == SKRenderAction.ActionTypes.Transform)
01770                 {
01771                     SKMatrix mat = ctx.SKRenderActions[i].Transform.Value;
01772                     canvas.Concat(ref mat);
01773                 }
01774                 else
01775                 {
01776                     if (ctx.SKRenderActions[i].ActionType == SKRenderAction.ActionTypes.Path &&
      ctx.SKRenderActions[i].Path != null)
01777                     {
01778                         canvas.DrawPath(ctx.SKRenderActions[i].Path, ctx.SKRenderActions[i].Paint);
01779                     }
01780                     else if (ctx.SKRenderActions[i].ActionType == SKRenderAction.ActionTypes.Text)
01781                     {
01782                         canvas.DrawText(ctx.SKRenderActions[i].Text, ctx.SKRenderActions[i].TextX,
      ctx.SKRenderActions[i].TextY, ctx.SKRenderActions[i].Font, ctx.SKRenderActions[i].Paint);
01783                     }
01784                     else if (ctx.SKRenderActions[i].ActionType ==
```

```
    SKRenderAction.ActionTypes.RasterImage)
01785                     {
01786                         (SKBitmap image, bool interpolateIt) =
    ctx.Images[ctx.SKRenderActions[i].ImageId];
01787
01788                         SKPaint paint;
01789
01790                         if (!interpolateIt)
01791                         {
01792                             paint = null;
01793                         }
01794                         else
01795                         {
01796                             paint = new SKPaint() { FilterQuality = SKFilterQuality.Medium };
01797                         }
01798
01799                         canvas.DrawBitmap(image, ctx.SKRenderActions[i].ImageSource.Value,
    ctx.SKRenderActions[i].ImageDestination.Value, paint);
01800
01801                         paint?.Dispose();
01802                     }
01803                 }
01804             }
01805
01806             canvas.Restore();
01807
01808             IntPtr pixels = bitmap.GetPixels(out IntPtr length);
01809
01810             IntPtr tbrData = System.Runtime.InteropServices.Marshal.AllocHGlobal(length);
01811             GC.AddMemoryPressure((long)length);
01812
01813             unsafe
01814             {
01815                 Buffer.MemoryCopy((void*)pixels, (void*)tbrData, (long)length, (long)length);
01816             }
01817
01818             canvas.Dispose();
01819             bitmap.Dispose();
01820
01821             DisposableIntPtr disp = new DisposableIntPtr(tbrData);
01822             return new RasterImage(ref disp, width, height, true, interpolate);
01823         }
01824     }
01825 }
```

8.7 Nimbus.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU General Public License as published by
00007 the Free Software Foundation, version 3.
00008 This program is distributed in the hope that it will be useful,
00009 but WITHOUT ANY WARRANTY; without even the implied warranty of
00010 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00011 GNU General Public License for more details.
00012 You should have received a copy of the GNU General Public License
00013 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00014 */
00015
00016 namespace VectSharp.Fonts
00017 {
00018 /// <summary>
00019 /// Contains an <see cref="IFontLibrary"/> providing access to the Nimbus family of standard fonts
    (used e.g.  by MuPDF).
00020 /// </summary>
00021     public class Nimbus
00022     {
00023 /// <summary>
00024 /// The font library.
00025 /// </summary>
00026         public static IFontLibrary Library { get; }
00027
00028         static Nimbus()
00029         {
00030             Library = new SimpleFontLibrary(
00031                 new
    ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.NimbusRomNo9L-Reg.t
    "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#Nimbus Roman No9 L"),
00032                 new
    ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.NimbusRomNo9L-Med.t
    "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#Nimbus Roman No9 L"),
```

```
00033                 new
      ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.NimbusRomNo9L-RegIta
      "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#Nimbus Roman No9 L"),
00034                 new
      ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.NimbusRomNo9L-MedIta
      "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#Nimbus Roman No9 L"),
00035                 new
      ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.NimbusSanL-Reg.ttf"
      "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#Nimbus Sans L"),
00036                 new
      ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.NimbusSanL-Bol.ttf"
      "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#Nimbus Sans L"),
00037                 new
      ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.NimbusSanL-RegIta.tt
      "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#Nimbus Sans L"),
00038                 new
      ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.NimbusSanL-BolIta.tt
      "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#Nimbus Sans L"),
00039                 new
      ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.NimbusMono-Regular.t
      "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#Nimbus Mono"),
00040                 new
      ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.NimbusMono-Bold.ttf"
      "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#Nimbus Mono"),
00041                 new
      ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.NimbusMono-Oblique.t
      "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#Nimbus Mono"),
00042                 new
      ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.NimbusMono-BoldOblic
      "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#Nimbus Mono"),
00043                 new
      ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.StandardSymbolsPS.tt
      "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#StandardSymbolsPS"),
00044                 new
      ResourceFontFamily(typeof(Nimbus).Assembly.GetManifestResourceStream("VectSharp.Fonts.Nimbus.Nimbus.D050000L.ttf"),
      "resm:VectSharp.Fonts.Nimbus.Nimbus.?assembly=VectSharp.Fonts.Nimbus#D050000L"));
00045         }
00046     }
00047 }
```

8.8 HtmlTag.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using ExCSS;
00019 using System;
00020 using System.Collections.Generic;
00021 using System.IO;
00022 using System.Linq;
00023 using System.Net;
00024 using System.Text;
00025 using VectSharp.SVG;
00026
00027 namespace VectSharp.Markdown
00028 {
00029     internal class HtmlTag
00030     {
00031         public Dictionary<string, string> Attributes { get; }
00032         public string Tag { get; }
00033
00034         private HtmlTag(string tag, Dictionary<string, string> attributes)
00035         {
00036             this.Tag = tag;
00037             this.Attributes = attributes;
00038         }
00039
00040         private static HtmlTag ParseTag(StringReader reader)
00041         {
```

```
00042             StringBuilder tagBuilder = new StringBuilder();
00043
00044             int character = reader.Read();
00045
00046             while (character >= 0 && (char)character != '<')
00047             {
00048                 character = reader.Read();
00049             }
00050
00051             if ((char)character == '<')
00052             {
00053                 character = reader.Read();
00054             }
00055
00056             while (character >= 0 && char.IsWhiteSpace((char)character))
00057             {
00058                 character = reader.Read();
00059             }
00060
00061             while (character >= 0 && !char.IsWhiteSpace((char)character) && (char)character != '>')
00062             {
00063                 tagBuilder.Append((char)character);
00064                 character = reader.Read();
00065             }
00066
00067             string tag = tagBuilder.ToString();
00068
00069             Dictionary<string, string> attributes = new Dictionary<string, string>();
00070
00071             (string, string)?  attribute = ReadAttribute(reader, ref character);
00072
00073             while (attribute != null && character >= 0)
00074             {
00075                 attributes[attribute.Value.Item1] = attribute.Value.Item2;
00076                 attribute = ReadAttribute(reader, ref character);
00077             }
00078
00079             return new HtmlTag(tag, attributes);
00080         }
00081
00082         public static IEnumerable<HtmlTag> ParseTagsUntil(StringReader reader, string targetTag)
00083         {
00084             HtmlTag tag = ParseTag(reader);
00085
00086             while (tag.Tag != targetTag && reader.Peek() >= 0)
00087             {
00088                 if (tag.Tag.Equals("p", StringComparison.OrdinalIgnoreCase))
00089                 {
00090                     foreach (HtmlTag nestedTag in ParseTagsUntil(reader, "/p"))
00091                     {
00092                         if (nestedTag.Tag != "/p")
00093                         {
00094                             foreach (KeyValuePair<string, string> kvp in tag.Attributes)
00095                             {
00096                                 if (!nestedTag.Attributes.ContainsKey(kvp.Key))
00097                                 {
00098                                     nestedTag.Attributes[kvp.Key] = kvp.Value;
00099                                 }
00100                             }
00101
00102                             yield return nestedTag;
00103                         }
00104                     }
00105                 }
00106                 else
00107                 {
00108                     yield return tag;
00109                 }
00110
00111
00112                 tag = ParseTag(reader);
00113             }
00114
00115             yield return tag;
00116         }
00117
00118         public static IEnumerable<HtmlTag> Parse(string html)
00119         {
00120             using (StringReader reader = new StringReader(html))
00121             {
00122                 while (reader.Peek() >= 0)
00123                 {
00124                     HtmlTag tag = ParseTag(reader);
00125
00126                     if (tag.Tag.Equals("p", StringComparison.OrdinalIgnoreCase))
00127                     {
00128                         foreach (HtmlTag nestedTag in ParseTagsUntil(reader, "/p"))
```

```
00129                         {
00130                             if (nestedTag.Tag != "/p")
00131                             {
00132                                 foreach (KeyValuePair<string, string> kvp in tag.Attributes)
00133                                 {
00134                                     if (!nestedTag.Attributes.ContainsKey(kvp.Key))
00135                                     {
00136                                         nestedTag.Attributes[kvp.Key] = kvp.Value;
00137                                     }
00138                                 }
00139 
00140                                 yield return nestedTag;
00141                             }
00142                         }
00143                     }
00144                     else
00145                     {
00146                         yield return tag;
00147                     }
00148                 }
00149             }
00150         }
00151 
00152         private static (string, string)?  ReadAttribute(StringReader reader, ref int character)
00153         {
00154             while (character >= 0 && char.IsWhiteSpace((char)character) && (char)character != '>')
00155             {
00156                 character = reader.Read();
00157             }
00158 
00159             if ((char)character == '>')
00160             {
00161                 return null;
00162             }
00163             else
00164             {
00165                 StringBuilder attributeNameBuilder = new StringBuilder();
00166 
00167                 while (character >= 0 && !char.IsWhiteSpace((char)character) && (char)character != '>'
      && (char)character != '=')
00168                 {
00169                     attributeNameBuilder.Append((char)character);
00170                     character = reader.Read();
00171                 }
00172 
00173                 string attributeName = attributeNameBuilder.ToString();
00174 
00175                 while (character >= 0 && char.IsWhiteSpace((char)character) && (char)character != '>'
      && (char)character != '=')
00176                 {
00177                     character = reader.Read();
00178                 }
00179 
00180                 if ((char)character == '=')
00181                 {
00182                     character = reader.Read();
00183 
00184                     while (character >= 0 && char.IsWhiteSpace((char)character) && (char)character !=
      '>')
00185                     {
00186                         character = reader.Read();
00187                     }
00188 
00189                     if ((char)character == '>')
00190                     {
00191                         return (attributeName, null);
00192                     }
00193                     else
00194                     {
00195                         bool quoted = (char)character == '"' || (char)character == '\'';
00196 
00197                         if (quoted)
00198                         {
00199                             char quoteChar = (char)character;
00200 
00201                             character = reader.Read();
00202 
00203                             StringBuilder attributeValueBuilder = new StringBuilder();
00204 
00205                             bool isEscaped = (char)character == '\\';
00206 
00207                             while (character >= 0 && ((char)character != quoteChar || isEscaped))
00208                             {
00209                                 attributeValueBuilder.Append((char)character);
00210                                 character = reader.Read();
00211                                 isEscaped = (char)character == '\\' && !isEscaped;
00212                             }
```

```
00213
00214                               string attributeValue = attributeValueBuilder.ToString();
00215
00216                               return (attributeName, attributeValue);
00217                           }
00218                           else
00219                           {
00220                               StringBuilder attributeValueBuilder = new StringBuilder();
00221
00222                               while (character >= 0 && !char.IsWhiteSpace((char)character) &&
      (char)character != '>' && (char)character != '=')
00223                               {
00224                                   attributeValueBuilder.Append((char)character);
00225                                   character = reader.Read();
00226                               }
00227
00228                               string attributeValue = attributeValueBuilder.ToString();
00229
00230                               return (attributeName, attributeValue);
00231                           }
00232                       }
00233                   }
00234                   else
00235                   {
00236                       return (attributeName, null);
00237                   }
00238               }
00239           }
00240
00241       }
00242
00243 /// <summary>
00244 /// Contains utilities to resolve absolute and relative URIs.
00245 /// </summary>
00246       public static class HTTPUtils
00247       {
00248 /// <summary>
00249 /// Determines whether every file that is downloaded should be logged to the standard error stream.
00250 /// </summary>
00251           public static bool LogDownloads { get; set; } = true;
00252
00253 /// <summary>
00254 /// Uses OpenMoji (https://openmoji.org) to resolve emoji URIs of the form
      "emoji://{name}_heading:{level}" or "unicode://{unicode}_heading:{level}".
00255 /// </summary>
00256 /// <param name="emojiUri">The emoji URI.</param>
00257 /// <param name="renderer">The <see cref="MarkdownRenderer"/> on which the emoji will be
      drawn.</param>
00258 /// <returns>A tuple containing the local path of the rendered emoji and a boolean value indicating
      whether the image should be deleted after the program is done with it.</returns>
00259           public static (string path, bool wasDownloaded) ResolveEmojiUsingOpenMoji(string emojiUri,
      MarkdownRenderer renderer)
00260           {
00261               int headingLevel = int.Parse(emojiUri.Substring(emojiUri.LastIndexOf("_heading:") + 9));
00262               string emojiUnicode;
00263
00264               if (emojiUri.StartsWith("emoji://"))
00265               {
00266                   string emojiName = emojiUri.Substring(8, emojiUri.LastIndexOf("_heading:") - 8);
00267                   emojiUnicode =
      Markdig.Extensions.Emoji.EmojiMapping.GetDefaultEmojiShortcodeToUnicode()[":" + emojiName + ":"];
00268               }
00269               else if (emojiUri.StartsWith("unicode://"))
00270               {
00271                   emojiUnicode = emojiUri.Substring(10, emojiUri.LastIndexOf("_heading:") - 10);
00272               }
00273               else
00274               {
00275                   return (null, false);
00276               }
00277
00278               // Adapted from OpenMoji FAQ (https://openmoji.org/faq/)
00279               string emojiCode = emojiUnicode.SelectMany((x, i) => !char.IsLowSurrogate(emojiUnicode, i)
      ?  new string[] { char.ConvertToUtf32(emojiUnicode, i).ToString("X4") } :  new
      string[0]).Aggregate((a, b) => a + "-" + b);
00280               if (emojiCode.Length == 10)
00281               {
00282                   emojiCode = emojiCode.Replace("-FE0F", "");
00283               }
00284               string url = "https://openmoji.org/data/color/svg/" + emojiCode + ".svg";
00285
00286               (string, bool) downloadedEmoji = renderer.ImageUriResolver(url, "");
00287
00288               if (string.IsNullOrEmpty(downloadedEmoji.Item1))
00289               {
00290                   string tempFile = Path.GetTempFileName();
00291                   File.Delete(tempFile);
```

```
00292                 Directory.CreateDirectory(tempFile);
00293                 string fileDest = Path.Combine(tempFile, "fail.svg");
00294 
00295                 using (FileStream fs = File.Create(fileDest))
00296                 {
00297                     using (Stream s =
    System.Reflection.Assembly.GetExecutingAssembly().GetManifestResourceStream("VectSharp.Markdown.Resources.MissingCodepo
00298                     {
00299                         s.CopyTo(fs);
00300                     }
00301                 }
00302 
00303                 downloadedEmoji = (fileDest, true);
00304             }
00305 
00306             Page emojiPage = VectSharp.SVG.Parser.FromFile(downloadedEmoji.Item1);
00307             double targetEmojiHeight = renderer.BaseFontSize * (headingLevel == 0 ?  1 :
    renderer.HeaderFontSizeMultipliers[headingLevel - 1]);
00308             Page scaledEmojiPage = new Page(emojiPage.Width * targetEmojiHeight / emojiPage.Height,
    targetEmojiHeight);
00309             scaledEmojiPage.Graphics.Scale(targetEmojiHeight / emojiPage.Height, targetEmojiHeight /
    emojiPage.Height);
00310             double y = -renderer.RegularFontFamily.TrueTypeFile.Get1000EmDescent() / 1000 *
    emojiPage.Height * 0.5;
00311             scaledEmojiPage.Graphics.DrawGraphics(0, y, emojiPage.Graphics);
00312 
00313             if (downloadedEmoji.Item2)
00314             {
00315                 scaledEmojiPage.SaveAsSVG(downloadedEmoji.Item1);
00316                 return downloadedEmoji;
00317             }
00318             else
00319             {
00320                 string tempFile = Path.GetTempFileName();
00321                 File.Delete(tempFile);
00322                 Directory.CreateDirectory(tempFile);
00323                 string fileDest = Path.Combine(tempFile,
    Path.GetFileNameWithoutExtension(downloadedEmoji.Item1) + ".svg");
00324                 scaledEmojiPage.SaveAsSVG(fileDest);
00325                 return (fileDest, true);
00326             }
00327         }
00328 
00329 /// <summary>
00330 /// Resolves an image Uri, by downloading the image file if necessary.  It also takes care of ensuring
    that the file extension matches the format of the file.
00331 /// </summary>
00332 /// <param name="uri">The address of the image.</param>
00333 /// <param name="baseUriString">The base uri to use for relative uris.</param>
00334 /// <returns>A tuple containing the local path of the image file (either the original image, or a
    local copy of a remote file) and a boolean value indicating whether the image was fetched from a
    remote location and should be deleted after the program is done with it.</returns>
00335         public static (string path, bool wasDownloaded) ResolveImageURI(string uri, string
    baseUriString)
00336         {
00337             if (uri.StartsWith("data:"))
00338             {
00339                 string tempFile = Path.GetTempFileName();
00340                 if (File.Exists(tempFile))
00341                 {
00342                     File.Delete(tempFile);
00343                 }
00344 
00345                 Directory.CreateDirectory(tempFile);
00346 
00347                 if (!uri.StartsWith(""))
00348                 {
00349                     VectSharp.Page pag = VectSharp.SVG.Parser.ParseImageURI(uri, true);
00350                     VectSharp.SVG.SVGContextInterpreter.SaveAsSVG(pag, Path.Combine(tempFile,
    "temp.svg"));
00351                 }
00352                 else
00353                 {
00354                     string base64 = uri.Substring("".Length);
00355 
00356                     File.WriteAllBytes(Path.Combine(tempFile, "temp.svg"),
    Convert.FromBase64String(base64));
00357                 }
00358 
00359                 return (Path.Combine(tempFile, "temp.svg"), true);
00360             }
00361             else if (File.Exists(Path.Combine(baseUriString, uri)))
00362             {
00363                 return (Path.Combine(baseUriString, uri), false);
00364             }
00365             else if (File.Exists(uri))
00366             {
```

```
00367                 return (uri, false);
00368             }
00369             else
00370             {
00371                 Uri absoluteUri;
00372                 bool validUri;
00373 
00374                 if (Uri.TryCreate(baseUriString, UriKind.Absolute, out Uri baseUri))
00375                 {
00376                     validUri = Uri.TryCreate(baseUri, uri, out absoluteUri);
00377                 }
00378                 else
00379                 {
00380                     validUri = Uri.TryCreate(uri, UriKind.Absolute, out absoluteUri);
00381                 }
00382 
00383                 if (validUri)
00384                 {
00385                     string tempFile = Path.GetTempFileName();
00386                     File.Delete(tempFile);
00387                     Directory.CreateDirectory(tempFile);
00388 
00389                     string fileDest = Path.Combine(tempFile, Path.GetFileName(absoluteUri.LocalPath));
00390 
00391                     try
00392                     {
00393                         if (LogDownloads)
00394                         {
00395                             Console.Error.WriteLine();
00396                             Console.Error.Write("Downloading {0}...", absoluteUri);
00397                         }
00398 
00399                         using (WebClient client = new WebClient())
00400                         {
00401                             client.DownloadFile(absoluteUri, fileDest);
00402                         }
00403 
00404                         if (LogDownloads)
00405                         {
00406                             Console.Error.WriteLine(" Done.");
00407                         }
00408 
00409                         string newName = FixFileExtensionBasedOnContent(fileDest);
00410 
00411                         File.Move(fileDest, newName);
00412                         fileDest = newName;
00413 
00414                         return (fileDest, true);
00415                     }
00416                     catch (Exception ex)
00417                     {
00418                         if (LogDownloads)
00419                         {
00420                             Console.Error.WriteLine(" Failed!");
00421                             Console.Error.WriteLine(ex.Message);
00422                         }
00423 
00424                         Directory.Delete(tempFile, true);
00425                         return (null, false);
00426                     }
00427                 }
00428                 else
00429                 {
00430                     return (null, false);
00431                 }
00432             }
00433         }
00434 
00435         private static string FixFileExtensionBasedOnContent(string fileName)
00436         {
00437             using (FileStream fileStream = File.OpenRead(fileName))
00438             {
00439                 bool isSvg = false;
00440 
00441                 try
00442                 {
00443                     using (var xmlReader = System.Xml.XmlReader.Create(fileStream))
00444                     {
00445                         isSvg = xmlReader.MoveToContent() == System.Xml.XmlNodeType.Element &&
    "svg".Equals(xmlReader.Name, StringComparison.OrdinalIgnoreCase);
00446                     }
00447                 }
00448                 catch
00449                 {
00450                     isSvg = false;
00451                 }
00452 
```

```
00453                 if (isSvg)
00454                 {
00455                     return fileName + ".svg";
00456                 }
00457                 else
00458                 {
00459                     fileStream.Seek(0, SeekOrigin.Begin);
00460                     byte[] header = new byte[8];
00461 
00462                     for (int i = 0; i < header.Length; i++)
00463                     {
00464                         header[i] = (byte)fileStream.ReadByte();
00465                     }
00466 
00467                     if (header[0] == 0x42 && header[1] == 0x4D)
00468                     {
00469                         return fileName + ".bmp";
00470                     }
00471                     else if (header[0] == 0x47 && header[1] == 0x49 && header[2] == 0x46 && header[3]
      == 0x38)
00472                     {
00473                         return fileName + ".gif";
00474                     }
00475                     else if (header[0] == 0xFF && header[1] == 0xD8 && header[2] == 0xFF && (header[3]
      == 0xDB || header[3] == 0xE0 || header[3] == 0xEE || header[3] == 0xE1))
00476                     {
00477                         return fileName + ".jpg";
00478                     }
00479                     else if (header[0] == 0x25 && header[1] == 0x50 && header[2] == 0x44 && header[3]
      == 0x46 && header[4] == 0x2D)
00480                     {
00481                         return fileName + ".pdf";
00482                     }
00483                     else if (header[0] == 0x89 && header[1] == 0x50 && header[2] == 0x4E && header[3]
      == 0x47 && header[4] == 0x0D && header[5] == 0x0A && header[6] == 0x1A && header[7] == 0x0A)
00484                     {
00485                         return fileName + ".png";
00486                     }
00487                     else if ((header[0] == 0x49 && header[1] == 0x49 && header[2] == 0x2A && header[3]
      == 0x00) || (header[0] == 0x4D && header[1] == 0x4D && header[2] == 0x00 && header[3] == 0x2A))
00488                     {
00489                         return fileName + ".tif";
00490                     }
00491                     else
00492                     {
00493                         return fileName;
00494                     }
00495                 }
00496             }
00497         }
00498     }
00499 }
```

8.9 Line.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
00019 using System.Collections.Generic;
00020 
00021 namespace VectSharp.Markdown
00022 {
00023 /// <summary>
00024 /// Represents one of the fragments that make up a <see cref="Line"/>.
00025 /// </summary>
00026     public abstract class LineFragment
00027     {
00028 /// <summary>
```

```
00029 /// Applies the specified translation to the line fragment.
00030 /// </summary>
00031 /// <param name="deltaX">The translation in the horizontal direction.</param>
00032 /// <param name="deltaY">The translation in the vertical direction.</param>
00033         public abstract void Translate(double deltaX, double deltaY);
00034
00035 /// <summary>
00036 /// Graphics action tag for the fragment.
00037 /// </summary>
00038         public string Tag { get; protected set; }
00039
00040 /// <summary>
00041 /// Get the ascent of the <see cref="LineFragment"/>.
00042 /// </summary>
00043 /// <param name="lineAscent">The ascent of the <see cref="Line"/> on which the fragment should be
      drawn.</param>
00044 /// <returns>The ascent of the <see cref="LineFragment"/>.</returns>
00045         public abstract double GetAscent(double lineAscent);
00046
00047 /// <summary>
00048 /// Get the vertical coordinate of the bottom of the <see cref="LineFragment"/>.
00049 /// </summary>
00050 /// <returns>The vertical coordinates of the bottom of the <see cref="LineFragment"/>.</returns>
00051         public abstract double GetMaxY();
00052
00053 /// <summary>
00054 /// Get the horizontal coordinate of the right side of the <see cref="LineFragment"/>.
00055 /// </summary>
00056 /// <returns>The horizontal coordinate of the right side of the <see cref="LineFragment"/>.</returns>
00057         public abstract double GetMaxX();
00058
00059 /// <summary>
00060 /// Render the <see cref="LineFragment"/> on the specified <paramref name="graphics"/> surface.
00061 /// </summary>
00062 /// <param name="graphics">The <see cref="Graphics"/> surface on which the <see cref="LineFragment"/>
      will be drawn.</param>
00063 /// <param name="deltaY">Additional translation on the vertical axis.</param>
00064 /// <returns>If a text element has been drawn, this method should return the bottom-right corner of
      the drawn text element, otherwise <see langword="null"/>.  This is used to update the <see
      cref="MarkdownContext.BottomRight"/> property.</returns>
00065         public abstract Point?  Render(Graphics graphics, double deltaY);
00066     }
00067
00068 /// <summary>
00069 /// A fragment containing text.
00070 /// </summary>
00071     public class TextFragment :  LineFragment
00072     {
00073 /// <summary>
00074 /// The text contained in the fragment.
00075 /// </summary>
00076         public string Text { get; }
00077
00078 /// <summary>
00079 /// The position of the text, relative to the cursor at the start of the line.
00080 /// </summary>
00081         public Point Position { get; private set; }
00082
00083 /// <summary>
00084 /// The font used to draw the text.
00085 /// </summary>
00086         public Font Font { get; }
00087
00088 /// <summary>
00089 /// The colour used to draw the text.
00090 /// </summary>
00091         public Colour Colour { get; }
00092
00093 /// <summary>
00094 /// Create a new <see cref="TextFragment"/>.
00095 /// </summary>
00096 /// <param name="position">The position of the text.</param>
00097 /// <param name="text">The text contained in the fragment.</param>
00098 /// <param name="font">The font used to draw the text.</param>
00099 /// <param name="colour">The colour used to draw the text.</param>
00100 /// <param name="tag">Graphics action tag for the text.</param>
00101         public TextFragment(Point position, string text, Font font, Colour colour, string tag)
00102         {
00103             this.Position = position;
00104             this.Text = text;
00105             this.Font = font;
00106             this.Colour = colour;
00107             this.Tag = tag;
00108         }
00109
00110 /// <inheritdoc/>
00111         public override void Translate(double deltaX, double deltaY)
```

```
00112         {
00113             this.Position = new Point(this.Position.X + deltaX, this.Position.Y + deltaY);
00114         }
00115 
00116 /// <inheritdoc/>
00117         public override double GetAscent(double lineAscent)
00118         {
00119             return this.Font.Ascent;
00120         }
00121 
00122 /// <inheritdoc/>
00123         public override double GetMaxY()
00124         {
00125             return this.Position.Y - this.Font.Descent;
00126         }
00127 
00128 /// <inheritdoc/>
00129         public override double GetMaxX()
00130         {
00131             Size size = this.Font.MeasureText(this.Text);
00132             return size.Width + this.Position.X;
00133         }
00134 
00135 /// <inheritdoc/>
00136         public override Point?  Render(Graphics graphics, double deltaY)
00137         {
00138             Size size = this.Font.MeasureText(this.Text);
00139             graphics.FillText(this.Position.X, this.Position.Y + deltaY, this.Text, this.Font,
      this.Colour, TextBaselines.Baseline, tag:  this.Tag);
00140             return new Point(size.Width + this.Position.X, this.Position.Y + size.Height + deltaY);
00141         }
00142     }
00143 
00144 /// <summary>
00145 /// Represents a text underline.
00146 /// </summary>
00147     public class UnderlineFragment :  LineFragment
00148     {
00149 /// <summary>
00150 /// The start position of the underline, relative to the cursor at the start of the line.
00151 /// </summary>
00152         public Point Start { get; private set; }
00153 
00154 /// <summary>
00155 /// The end position of the underline, relative to the cursor at the start of the line.
00156 /// </summary>
00157         public Point End { get; private set; }
00158 
00159 /// <summary>
00160 /// The colour of the underline.
00161 /// </summary>
00162         public Colour Colour { get; }
00163 
00164 /// <summary>
00165 /// The thickness of the underline.
00166 /// </summary>
00167         public double Thickness { get; }
00168 
00169 /// <summary>
00170 /// Create a new <see cref="UnderlineFragment"/>.
00171 /// </summary>
00172 /// <param name="start">The start position of the underline.</param>
00173 /// <param name="end">The end position of the underline.</param>
00174 /// <param name="colour">The colour of the underline.</param>
00175 /// <param name="thickness">The thickness of the underline.</param>
00176 /// <param name="tag">Graphics action tag for the underline.</param>
00177         public UnderlineFragment(Point start, Point end, Colour colour, double thickness, string tag)
00178         {
00179             this.Start = start;
00180             this.End = end;
00181             this.Colour = colour;
00182             this.Thickness = thickness;
00183             this.Tag = tag;
00184         }
00185 
00186 /// <inheritdoc/>
00187         public override void Translate(double deltaX, double deltaY)
00188         {
00189             this.Start = new Point(this.Start.X + deltaX, this.Start.Y + deltaY);
00190             this.End = new Point(this.End.X + deltaX, this.End.Y + deltaY);
00191         }
00192 
00193 /// <inheritdoc/>
00194         public override double GetAscent(double lineAscent)
00195         {
00196             return lineAscent;
00197         }
```

```
00198
00199 /// <inheritdoc/>
00200         public override double GetMaxY()
00201         {
00202             return Math.Max(this.Start.Y, this.End.Y);
00203         }
00204
00205 /// <inheritdoc/>
00206         public override double GetMaxX()
00207         {
00208             return Math.Max(this.Start.X, this.End.X);
00209         }
00210
00211 /// <inheritdoc/>
00212         public override Point?  Render(Graphics graphics, double deltaY)
00213         {
00214             graphics.StrokePath(new GraphicsPath().MoveTo(this.Start.X, this.Start.Y +
      deltaY).LineTo(this.End.X, this.End.Y + deltaY), this.Colour, this.Thickness, tag:  this.Tag);
00215             return null;
00216         }
00217     }
00218
00219 /// <summary>
00220 /// Represents a rectangle.
00221 /// </summary>
00222     public class RectangleFragment :  LineFragment
00223     {
00224 /// <summary>
00225 /// The top-left corner of the rectangle, relative to the cursor at the start of the line.
00226 /// </summary>
00227         public Point TopLeft { get; private set; }
00228
00229 /// <summary>
00230 /// The size of the rectangle.
00231 /// </summary>
00232         public Size Size { get; }
00233
00234 /// <summary>
00235 /// The colour used to draw the rectangle.
00236 /// </summary>
00237         public Colour Colour { get; }
00238
00239 /// <summary>
00240 /// Create a new <see cref="RectangleFragment"/>.
00241 /// </summary>
00242 /// <param name="topLeft">The top-left corner of the rectangle.</param>
00243 /// <param name="size">The size of the rectangle.</param>
00244 /// <param name="colour">The colour used to draw the rectangle.</param>
00245 /// <param name="tag">Graphics action tag for the rectangle.</param>
00246         public RectangleFragment(Point topLeft, Size size, Colour colour, string tag)
00247         {
00248             this.TopLeft = topLeft;
00249             this.Size = size;
00250             this.Colour = colour;
00251             this.Tag = tag;
00252         }
00253
00254 /// <inheritdoc/>
00255         public override void Translate(double deltaX, double deltaY)
00256         {
00257             this.TopLeft = new Point(this.TopLeft.X + deltaX, this.TopLeft.Y + deltaY);
00258         }
00259
00260 /// <inheritdoc/>
00261         public override double GetAscent(double lineAscent)
00262         {
00263             return lineAscent;
00264         }
00265
00266 /// <inheritdoc/>
00267         public override double GetMaxY()
00268         {
00269             return this.TopLeft.Y + this.Size.Height;
00270         }
00271
00272 /// <inheritdoc/>
00273         public override double GetMaxX()
00274         {
00275             return this.TopLeft.X + this.Size.Width;
00276         }
00277
00278 /// <inheritdoc/>
00279         public override Point?  Render(Graphics graphics, double deltaY)
00280         {
00281             graphics.FillRectangle(this.TopLeft.X, this.TopLeft.Y + deltaY, this.Size.Width,
      this.Size.Height, this.Colour, tag:  this.Tag);
00282             return null;
```

```
00283         }
00284     }
00285
00286 /// <summary>
00287 /// Represents a <see cref="LineFragment"/> that draws something.
00288 /// </summary>
00289     internal class GraphicsFragment :  LineFragment
00290     {
00291 /// <summary>
00292 /// The origin of the graphics fragment being drawn, relative to the cursor at the start of the line.
00293 /// </summary>
00294         public Point Origin { get; private set; }
00295
00296 /// <summary>
00297 /// The <see cref="Graphics"/> to draw.
00298 /// </summary>
00299         public Graphics Graphics { get; }
00300
00301 /// <summary>
00302 /// The ascent for the graphics.
00303 /// </summary>
00304         public double Ascent { get; }
00305
00306 /// <summary>
00307 /// Create a new <see cref="GraphicsFragment"/>.
00308 /// </summary>
00309 /// <param name="origin">The origin of the graphics fragment being drawn.</param>
00310 /// <param name="graphics">The <see cref="Graphics"/> to draw.</param>
00311 /// <param name="ascent">The ascent for the graphics.</param>
00312         public GraphicsFragment(Point origin, Graphics graphics, double ascent)
00313         {
00314             this.Origin = origin;
00315             this.Graphics = graphics;
00316             this.Ascent = ascent;
00317         }
00318
00319 /// <inheritdoc/>
00320         public override void Translate(double deltaX, double deltaY)
00321         {
00322             this.Origin = new Point(this.Origin.X + deltaX, this.Origin.Y + deltaY);
00323         }
00324
00325 /// <inheritdoc/>
00326         public override double GetAscent(double lineAscent)
00327         {
00328             return this.Ascent;
00329         }
00330
00331 /// <inheritdoc/>
00332         public override double GetMaxY()
00333         {
00334             return this.Origin.Y;
00335         }
00336
00337 /// <inheritdoc/>
00338         public override double GetMaxX()
00339         {
00340             return this.Origin.X + this.Graphics.GetBounds().Location.X +
    this.Graphics.GetBounds().Size.Width;
00341         }
00342
00343 /// <inheritdoc/>
00344         public override Point?  Render(Graphics graphics, double deltaY)
00345         {
00346             graphics.DrawGraphics(this.Origin.X, this.Origin.Y + deltaY, this.Graphics);
00347             return null;
00348         }
00349     }
00350
00351 /// <summary>
00352 /// Represents a text line.
00353 /// </summary>
00354     public class Line
00355     {
00356 /// <summary>
00357 /// The fragments contained in the line.
00358 /// </summary>
00359         public List<LineFragment> Fragments { get; }
00360
00361 /// <summary>
00362 /// The initial ascent for text on the <see cref="Line"/>.
00363 /// </summary>
00364         public double InitialAscent { get; }
00365
00366 /// <summary>
00367 /// Create a new <see cref="Line"/>.
00368 /// </summary>
```

```
00369 /// <param name="initialAscent">The initial ascent for text on the <see cref="Line"/>.</param>
00370         public Line(double initialAscent)
00371         {
00372             this.InitialAscent = initialAscent;
00373             this.Fragments = new List<LineFragment>();
00374         }
00375 
00376         internal void Render(ref Graphics graphics, ref MarkdownContext context,
      MarkdownRenderer.NewPageAction newPageAction, double pageMaxY, double pageMaxX, MarkdownRenderer
      renderer)
00377         {
00378             double deltaY = 0;
00379             double maxY = 0;
00380             double maxX = 0;
00381 
00382             foreach (LineFragment fragment in this.Fragments)
00383             {
00384                 deltaY = Math.Max(deltaY, fragment.GetAscent(this.InitialAscent) - InitialAscent);
00385                 maxY = Math.Max(maxY, fragment.GetMaxY());
00386                 maxX = Math.Max(maxX, fragment.GetMaxX());
00387             }
00388 
00389             maxY += deltaY;
00390 
00391             double minY = context.Cursor.Y -(this.InitialAscent + deltaY);
00392 
00393             Line line = this;
00394 
00395             renderer.RaiseLineMeasured(ref context, ref graphics, ref line, pageMaxX, pageMaxY, maxX,
      minY, maxY, context.Cursor.Y);
00396 
00397             if (line != null && line.Fragments.Count > 0)
00398             {
00399                 deltaY = 0;
00400                 maxY = 0;
00401                 maxX = 0;
00402 
00403                 foreach (LineFragment fragment in line.Fragments)
00404                 {
00405                     deltaY = Math.Max(deltaY, fragment.GetAscent(line.InitialAscent) - InitialAscent);
00406                     maxY = Math.Max(maxY, fragment.GetMaxY());
00407                     maxX = Math.Max(maxX, fragment.GetMaxX());
00408                 }
00409 
00410                 maxY += deltaY;
00411 
00412                 if (maxY > pageMaxY)
00413                 {
00414                     double currCursY = context.Cursor.Y;
00415 
00416                     newPageAction(ref context, ref graphics);
00417 
00418                     double currDelta = deltaY;
00419 
00420                     context.Cursor = new Point(context.Cursor.X, context.Cursor.Y + line.InitialAscent
      + currDelta);
00421 
00422                     deltaY -= currCursY - context.Cursor.Y + currDelta;
00423 
00424                     context.Cursor = new Point(context.Cursor.X, context.Cursor.Y);
00425                 }
00426                 else
00427                 {
00428                     context.Cursor = new Point(context.Cursor.X, context.Cursor.Y + deltaY);
00429                 }
00430 
00431                 renderer.RaiseLineRendering(ref context, ref graphics, line, pageMaxX, pageMaxY, maxX,
      minY, maxY, context.Cursor.Y);
00432 
00433                 for (int i = 0; i < line.Fragments.Count; i++)
00434                 {
00435                     Point?  pt = line.Fragments[i].Render(graphics, deltaY);
00436 
00437                     if (pt != null)
00438                     {
00439                         context.BottomRight = new Point(Math.Max(context.BottomRight.X, pt.Value.X +
      context.Translation.X), Math.Max(context.BottomRight.Y, pt.Value.Y + context.Translation.Y));
00440                     }
00441                 }
00442 
00443                 renderer.RaiseLineRendered(ref context, ref graphics, line, pageMaxX, pageMaxY, maxX,
      minY, maxY, context.Cursor.Y);
00444             }
00445         }
00446     }
00447 }
```

8.10 MarkdownContext.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020
00021 namespace VectSharp.Markdown
00022 {
00023 /// <summary>
00024 /// Represents the current status of the Markdown rendering process.
00025 /// </summary>
00026     public class MarkdownContext
00027     {
00028 /// <summary>
00029 /// The current font.
00030 /// </summary>
00031         public Font Font { get; set; }
00032
00033         private Point cursor;
00034
00035 /// <summary>
00036 /// The current position of the cursor.
00037 /// </summary>
00038         public Point Cursor
00039         {
00040             get
00041             {
00042                 return cursor;
00043             }
00044
00045             set
00046             {
00047                 cursor = value;
00048
00049                 double maxX = Math.Max(BottomRight.X, value.X + Translation.X);
00050                 double maxY = Math.Max(BottomRight.Y, value.Y + Translation.Y);
00051
00052                 this.BottomRight = new Point(maxX, maxY);
00053             }
00054         }
00055
00056 /// <summary>
00057 /// The current text colour.
00058 /// </summary>
00059         public Colour Colour { get; set; }
00060
00061 /// <summary>
00062 /// Whether text is currently underlined.
00063 /// </summary>
00064         public bool Underline { get; set; }
00065
00066 /// <summary>
00067 /// Whether text is currently struck out.
00068 /// </summary>
00069         public bool StrikeThrough { get; set; }
00070
00071         private Point translation;
00072
00073 /// <summary>
00074 /// Translation for the current text block.
00075 /// </summary>
00076         public Point Translation
00077         {
00078             get
00079             {
00080                 return translation;
00081             }
00082
00083             set
00084             {
00085                 translation = value;
```

```
00086
00087                 double maxX = Math.Max(BottomRight.X, cursor.X + value.X);
00088                 double maxY = Math.Max(BottomRight.Y, cursor.Y + value.Y);
00089
00090                 this.BottomRight = new Point(maxX, maxY);
00091             }
00092         }
00093
00094 /// <summary>
00095 /// The coordinates of the bottom right corner of the last rendered text element.
00096 /// </summary>
00097         public Point BottomRight { get; set; }
00098
00099 /// <summary>
00100 /// The current margin on the bottom right.
00101 /// </summary>
00102         public Point MarginBottomRight { get; set; }
00103
00104 /// <summary>
00105 /// The current list depth (for bullet lists and numbered lists).
00106 /// </summary>
00107         public int ListDepth { get; set; }
00108
00109 /// <summary>
00110 /// A list of areas on which text cannot be drawn (because they contain a right-aligned image).
00111 /// Each element of this list specifies a maximum x value (first element) and a y range (second and
      third element).
00112 /// Within the specified y range, text cannot extend beyond the maximum x value.
00113 /// </summary>
00114         public List<(double MaxX, double MinY, double MaxY)> ForbiddenAreasRight { get; set; }
00115
00116 /// <summary>
00117 /// A list of areas on which text cannot be drawn (because they contain a left-aligned image).
00118 /// Each element of this list specifies a minimum x value (first element) and a y range (second and
      third element).
00119 /// Within the specified y range, text starts at the minimum x value.
00120 /// </summary>
00121         public List<(double MinX, double MinY, double MaxY)> ForbiddenAreasLeft { get; set; }
00122
00123 /// <summary>
00124 /// The <see cref="Page"/> on which the document is being rendered.
00125 /// </summary>
00126         public Page CurrentPage { get; set; }
00127
00128 /// <summary>
00129 /// The current tag.
00130 /// </summary>
00131         public string Tag { get; set; } = null;
00132
00133 /// <summary>
00134 /// A dictionary used to associate graphic action tags to hyperlinks.
00135 /// </summary>
00136         public Dictionary<string, string> LinkDestinations { get; set; } = new Dictionary<string,
      string>();
00137
00138 /// <summary>
00139 /// A dictionary used to associate internal html anchors (e.g., "#something") to graphics action tags.
00140 /// </summary>
00141         public Dictionary<string, string> InternalAnchors { get; set; } = new Dictionary<string,
      string>();
00142
00143 /// <summary>
00144 /// The list of headings defined in the document.
00145 /// </summary>
00146         public List<(int level, string heading, string tag)> Headings { get; private set; } = new
      List<(int level, string heading, string tag)>();
00147
00148 /// <summary>
00149 /// The current line being drawn.
00150 /// </summary>
00151         public Line CurrentLine { get; set; }
00152
00153 /// <summary>
00154 /// Create a new <see cref="MarkdownContext"/>.
00155 /// </summary>
00156         internal MarkdownContext()
00157         {
00158             this.ForbiddenAreasRight = new List<(double MaxX, double MinY, double MaxY)>();
00159             this.ForbiddenAreasLeft = new List<(double MinX, double MinY, double MaxY)>();
00160         }
00161
00162 /// <summary>
00163 /// Create an independent copy of the current <see cref="MarkdownContext"/>.
00164 /// </summary>
00165 /// <returns>The copied context.</returns>
00166         public MarkdownContext Clone()
00167         {
```

```
00168             return new MarkdownContext()
00169             {
00170                 Font = this.Font,
00171                 Cursor = this.Cursor,
00172                 Colour = this.Colour,
00173                 Underline = this.Underline,
00174                 StrikeThrough = this.StrikeThrough,
00175                 Translation = this.Translation,
00176                 CurrentLine = this.CurrentLine,
00177                 ListDepth = this.ListDepth,
00178                 ForbiddenAreasRight = this.ForbiddenAreasRight,
00179                 ForbiddenAreasLeft = this.ForbiddenAreasLeft,
00180                 CurrentPage = this.CurrentPage,
00181                 Tag = this.Tag,
00182                 LinkDestinations = this.LinkDestinations,
00183                 InternalAnchors = this.InternalAnchors,
00184                 MarginBottomRight = this.MarginBottomRight,
00185                 Headings = this.Headings,
00186             };
00187         }
00188
00189 /// <summary>
00190 /// Get the maximum x value for text drawn at the specified <paramref name="y"/> position.
00191 /// </summary>
00192 /// <param name="y">The vertical position at which text should be drawn.</param>
00193 /// <param name="pageMaxX">The overall maximum x value for the page, after taking margins into
      account.</param>
00194 /// <returns>The maximum x value for text drawn at the specified <paramref name="y"/>
      position.</returns>
00195         public double GetMaxX(double y, double pageMaxX)
00196         {
00197             y += Translation.Y;
00198
00199             double maxX = pageMaxX;
00200
00201             foreach ((double MaxX, double MinY, double MaxY) in ForbiddenAreasRight)
00202             {
00203                 if (MinY <= y && MaxY >= y)
00204                 {
00205                     maxX = Math.Min(maxX, MaxX - this.Translation.X);
00206                 }
00207             }
00208
00209             return maxX;
00210         }
00211
00212 /// <summary>
00213 /// Get the maximum x value for text that will extend from <paramref name="y0"/> to <paramref
      name="y1"/>.
00214 /// </summary>
00215 /// <param name="y0">The vertical position of the start of the text block.</param>
00216 /// <param name="y1">The vertical position of the end of the text block.</param>
00217 /// <param name="pageMaxX">The overall maximum x value for the page, after taking margins into
      account.</param>
00218 /// <returns>The maximum x value for text that will extend from <paramref name="y0"/> to <paramref
      name="y1"/>.</returns>
00219         public double GetMaxX(double y0, double y1, double pageMaxX)
00220         {
00221             y0 += Translation.Y;
00222             y1 += Translation.Y;
00223
00224             double maxX = pageMaxX;
00225
00226             foreach ((double MaxX, double MinY, double MaxY) in ForbiddenAreasRight)
00227             {
00228                 if (!((MinY < y0 && MaxY < y0) || (MinY > y1 && MaxY > y1)))
00229                 {
00230                     maxX = Math.Min(maxX, MaxX - this.Translation.X);
00231                 }
00232             }
00233
00234             return maxX;
00235         }
00236
00237 /// <summary>
00238 /// Get the minimum x value at which text can start at the specified <paramref name="y"/> position.
00239 /// </summary>
00240 /// <param name="y">The vertical position at which text should be drawn.</param>
00241 /// <returns>The minimum x value at which text can start at the specified <paramref name="y"/>
      position.</returns>
00242         public double GetMinX(double y)
00243         {
00244             y += Translation.Y;
00245
00246             double minX = 0;
00247
00248             foreach ((double MinX, double MinY, double MaxY) in ForbiddenAreasLeft)
```

```
00249             {
00250                 if (MinY <= y && MaxY >= y)
00251                 {
00252                     minX = Math.Max(minX, MinX - Translation.X);
00253                 }
00254             }
00255 
00256             return minX;
00257         }
00258 
00259 /// <summary>
00260 /// Get the minimum x value for text that will extend from <paramref name="y0"/> to <paramref
      name="y1"/>.
00261 /// </summary>
00262 /// <param name="y0">The vertical position of the start of the text block.</param>
00263 /// <param name="y1">The vertical position of the end of the text block.</param>
00264 /// <returns>The minimum x value for text that will extend from <paramref name="y0"/> to <paramref
      name="y1"/>.</returns>
00265         public double GetMinX(double y0, double y1)
00266         {
00267             y0 += Translation.Y;
00268             y1 += Translation.Y;
00269 
00270             double minX = 0;
00271 
00272             foreach ((double MinX, double MinY, double MaxY) in ForbiddenAreasLeft)
00273             {
00274                 if (!((MinY < y0 && MaxY < y0) || (MinY > y1 && MaxY > y1)))
00275                 {
00276                     minX = Math.Max(minX, MinX - Translation.X);
00277                 }
00278             }
00279 
00280             return minX;
00281         }
00282 
00283 /// <summary>
00284 /// Start a new page in the document.
00285 /// </summary>
00286 /// <param name="newPag">The new page on which the rest of the document should be drawn.</param>
00287         public void StartNewPage(Page newPag)
00288         {
00289             this.Cursor = new Point(0, 0);
00290             this.ForbiddenAreasLeft.Clear();
00291             this.ForbiddenAreasRight.Clear();
00292             this.CurrentPage = newPag;
00293         }
00294     }
00295 
00296 /// <summary>
00297 /// Represents the margins of a page.
00298 /// </summary>
00299     public class Margins
00300     {
00301 /// <summary>
00302 /// The left margin.
00303 /// </summary>
00304         public double Left { get; }
00305 
00306 /// <summary>
00307 /// The right margin.
00308 /// </summary>
00309         public double Right { get; }
00310 
00311 /// <summary>
00312 /// The top margin.
00313 /// </summary>
00314         public double Top { get; }
00315 
00316 /// <summary>
00317 /// The bottom margin.
00318 /// </summary>
00319         public double Bottom { get; }
00320 
00321 /// <summary>
00322 /// Creates a new <see cref="Margins"/> instance.
00323 /// </summary>
00324 /// <param name="left">The left margin.</param>
00325 /// <param name="top">The top margin.</param>
00326 /// <param name="right">The right margin.</param>
00327 /// <param name="bottom">The bottom margin.</param>
00328         public Margins(double left, double top, double right, double bottom)
00329         {
00330             this.Left = left;
00331             this.Right = right;
00332             this.Top = top;
00333             this.Bottom = bottom;
```

```
00334         }
00335     }
00336 }
```

8.11 MarkdownRenderer.Blocks.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2024 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using Markdig.Helpers;
00019 using Markdig.Renderers.Html;
00020 using Markdig.Syntax.Inlines;
00021 using Markdig.Syntax;
00022 using System.Collections.Generic;
00023 using System.IO;
00024 using System.Text;
00025 using System;
00026 using VectSharp.Markdown.CSharpMath.VectSharp;
00027 using System.Linq;
00028 using Markdig.Extensions.Tables;
00029
00030 namespace VectSharp.Markdown
00031 {
00032     public partial class MarkdownRenderer
00033     {
00034         private void RenderBlock(Block block, ref MarkdownContext context, ref Graphics graphics,
      NewPageAction newPageAction, bool spaceBefore, bool spaceAfter)
00035         {
00036             this.OnBlockRendering(ref context, ref graphics, ref block);
00037
00038             if (block != null)
00039             {
00040                 HtmlAttributes attributes = block.TryGetAttributes();
00041
00042                 string tag = null;
00043
00044                 if (attributes != null && !string.IsNullOrEmpty(attributes.Id))
00045                 {
00046                     Point cursor = context.Cursor;
00047
00048                     void reversibleNewPageAction(ref MarkdownContext currContext, ref Graphics
      currGraphics)
00049                     {
00050                         newPageAction(ref currContext, ref currGraphics);
00051                         cursor = currContext.Cursor;
00052                     }
00053
00054                     RenderHTMLBlock("<a name=\"" + attributes.Id + "\"></a>", false, ref context, ref
      graphics, reversibleNewPageAction, spaceBefore, spaceAfter);
00055                     tag = context.InternalAnchors["#" + attributes.Id];
00056                     context.Cursor = cursor;
00057                 }
00058
00059                 if (block is LeafBlock leaf)
00060                 {
00061                     if (leaf is HeadingBlock heading)
00062                     {
00063                         string headingText = RenderHeadingBlock(heading, ref context, ref graphics,
      newPageAction, spaceBefore, spaceAfter);
00064
00065                         context.Headings.Add((heading.Level, headingText, tag));
00066                     }
00067                     else if (leaf is ParagraphBlock paragraph)
00068                     {
00069                         RenderParagraphBlock(paragraph, ref context, ref graphics, newPageAction,
      spaceBefore, spaceAfter);
00070                     }
00071                     else if (leaf is CodeBlock code)
00072                     {
```

```
00073                         if (block is Markdig.Extensions.Mathematics.MathBlock math)
00074                         {
00075                             StringBuilder mathBuilder = new StringBuilder();
00076                             foreach (StringLine line in math.Lines)
00077                             {
00078                                 mathBuilder.Append(line.ToString());
00079                                 mathBuilder.Append("\n");
00080                             }
00081 
00082                             byte[] svgData;
00083 
00084                             using (MemoryStream ms = new MemoryStream())
00085                             {
00086                                 MathPainter.LineStyle = global::CSharpMath.Atom.LineStyle.Display;
00087                                 MathPainter.FontSize = (float)(context.Font.FontSize *
      MathFontScalingFactor / ImageMultiplier);
00088                                 MathPainter.LaTeX = mathBuilder.ToString();
00089                                 Page pag = MathPainter.DrawToPage();
00090                                 VectSharp.SVG.SVGContextInterpreter.SaveAsSVG(pag, ms);
00091                                 svgData = ms.ToArray();
00092                             }
00093 
00094                             string imageUri = "<img align=\"center\" src=\""
      + Convert.ToBase64String(svgData) + "\">";
00095                             RenderHTMLBlock(imageUri, false, ref context, ref graphics, newPageAction,
      true, true);
00096                         }
00097                         else if (leaf is FencedCodeBlock fenced)
00098                         {
00099                             if (!string.IsNullOrEmpty(fenced.Info))
00100                             {
00101                                 RenderFencedCodeBlock(fenced, ref context, ref graphics,
      newPageAction, spaceBefore, spaceAfter);
00102                             }
00103                             else
00104                             {
00105                                 RenderCodeBlock(code, ref context, ref graphics, newPageAction,
      spaceBefore, spaceAfter);
00106                             }
00107 
00108                         }
00109                         else
00110                         {
00111                             RenderCodeBlock(code, ref context, ref graphics, newPageAction,
      spaceBefore, spaceAfter);
00112                         }
00113                     }
00114                     else if (leaf is HtmlBlock html)
00115                     {
00116                         RenderHTMLBlock(html.Lines.ToString(), false, ref context, ref graphics,
      newPageAction, spaceBefore, spaceAfter);
00117                     }
00118                     else if (leaf is ThematicBreakBlock thematicBreak)
00119                     {
00120                         RenderThematicBreakBlock(thematicBreak, ref context, ref graphics,
      newPageAction, spaceBefore, spaceAfter);
00121                     }
00122                     else if (leaf is LinkReferenceDefinition link)
00123                     {
00124                         // Nothing to do (the links are correctly referenced by the parser)
00125                     }
00126 
00127                 }
00128                 else if (block is ContainerBlock)
00129                 {
00130                     if (block is ListBlock list)
00131                     {
00132                         RenderListBlock(list, ref context, ref graphics, newPageAction);
00133                     }
00134                     else if (block is ListItemBlock listItem)
00135                     {
00136                         RenderListItemBlock(listItem, ref context, ref graphics, newPageAction);
00137                     }
00138                     else if (block is QuoteBlock quote)
00139                     {
00140                         RenderQuoteBlock(quote, ref context, ref graphics, newPageAction);
00141                     }
00142                     else if (block is LinkReferenceDefinitionGroup linkGroup)
00143                     {
00144                         // Nothing to render here
00145                     }
00146                     else if (block is Table table)
00147                     {
00148                         RenderTable(table, ref context, ref graphics, newPageAction);
00149                     }
00150                 }
00151                 else if (block is BlankLineBlock)
```

```
00152                 {
00153                     // Nothing to render here
00154                 }
00155 
00156                 this.OnBlockRendered(ref context, ref graphics, block);
00157             }
00158         }
00159 
00160         private string RenderHeadingBlock(HeadingBlock heading, ref MarkdownContext context, ref
      Graphics graphics, NewPageAction newPageAction, bool spaceBefore, bool spaceAfter)
00161         {
00162             MarkdownContext prevContext = context.Clone();
00163 
00164             context.Font = new Font(this.RegularFontFamily, BaseFontSize *
      HeaderFontSizeMultipliers[heading.Level - 1]);
00165 
00166             double minX = context.GetMinX(context.Cursor.Y + SpaceBeforeHeading *
      context.Font.FontSize, context.Cursor.Y + context.Font.Ascent + SpaceBeforeHeading *
      context.Font.FontSize - context.Font.Descent);
00167 
00168             if (spaceBefore)
00169             {
00170                 context.Cursor = new Point(minX, context.Cursor.Y + context.Font.Ascent +
      SpaceBeforeHeading * context.Font.FontSize);
00171             }
00172             else
00173             {
00174                 context.Cursor = new Point(minX, context.Cursor.Y + context.Font.Ascent);
00175             }
00176 
00177             if (context.CurrentLine == null)
00178             {
00179                 context.CurrentLine = new Line(context.Font.Ascent);
00180             }
00181             else
00182             {
00183                 double delta = context.Cursor.Y - (prevContext.Cursor.Y + prevContext.Font.Ascent +
      SpaceBeforeParagaph * prevContext.Font.FontSize);
00184 
00185                 for (int i = 0; i < context.CurrentLine.Fragments.Count; i++)
00186                 {
00187                     context.CurrentLine.Fragments[i].Translate(0, delta);
00188                 }
00189             }
00190 
00191             StringBuilder headingText = new StringBuilder();
00192 
00193             foreach (Inline inline in heading.Inline)
00194             {
00195                 headingText.Append(RenderInline(inline, ref context, ref graphics, newPageAction));
00196             }
00197 
00198             if (this.HeaderLineThicknesses[heading.Level - 1] > 0)
00199             {
00200                 double lineY = context.Cursor.Y + context.Font.FontSize * 0.3;
00201                 context.CurrentLine.Fragments.Add(new UnderlineFragment(new Point(minX,
      context.Cursor.Y + context.Font.FontSize * 0.3), new Point(context.GetMaxX(lineY, PageSize.Width -
      Margins.Right - context.Translation.X), lineY), this.HeaderLineColour,
      this.HeaderLineThicknesses[heading.Level - 1], context.Tag));
00202             }
00203 
00204             context.CurrentLine.Render(ref graphics, ref context, newPageAction, this.PageSize.Height
      - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y,
      context.GetMaxX(context.Cursor.Y - context.Font.Ascent, context.Cursor.Y - context.Font.Descent,
      this.PageSize.Width - this.Margins.Right - context.Translation.X - context.MarginBottomRight.X),
      this);
00205             context.CurrentLine = null;
00206 
00207             if (this.HeaderLineThicknesses[heading.Level - 1] > 0)
00208             {
00209                 double lineY = context.Cursor.Y + context.Font.FontSize * 0.3;
00210                 context.Cursor = new Point(context.Cursor.X, lineY +
      this.HeaderLineThicknesses[heading.Level - 1]);
00211             }
00212 
00213             context.Cursor = new Point(0, context.Cursor.Y - context.Font.Descent + SpaceAfterLine *
      context.Font.FontSize);
00214 
00215             if (spaceAfter)
00216             {
00217                 context.Cursor = new Point(0, context.Cursor.Y + SpaceAfterHeading *
      context.Font.FontSize);
00218             }
00219 
00220             prevContext.Cursor = context.Cursor;
00221             prevContext.BottomRight = context.BottomRight;
00222             prevContext.CurrentPage = context.CurrentPage;
```

```
00223             prevContext.CurrentLine = context.CurrentLine;
00224
00225             context = prevContext;
00226
00227             return headingText.ToString();
00228         }
00229
00230         private void RenderParagraphBlock(ParagraphBlock paragraph, ref MarkdownContext context, ref
      Graphics graphics, NewPageAction newPageAction, bool spaceBefore, bool spaceAfter)
00231         {
00232             double minX = context.GetMinX(context.Cursor.Y + SpaceBeforeParagaph *
      context.Font.FontSize, context.Cursor.Y + context.Font.Ascent + SpaceBeforeParagaph *
      context.Font.FontSize - context.Font.Descent);
00233
00234             if (spaceBefore)
00235             {
00236                 context.Cursor = new Point(minX, context.Cursor.Y + context.Font.Ascent +
      SpaceBeforeParagaph * context.Font.FontSize);
00237             }
00238             else
00239             {
00240                 context.Cursor = new Point(minX, context.Cursor.Y + context.Font.Ascent);
00241             }
00242
00243             context.CurrentLine ??= new Line(context.Font.Ascent);
00244
00245             foreach (Inline inline in paragraph.Inline)
00246             {
00247                 RenderInline(inline, ref context, ref graphics, newPageAction);
00248             }
00249
00250             context.CurrentLine.Render(ref graphics, ref context, newPageAction, this.PageSize.Height
      - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y,
      context.GetMaxX(context.Cursor.Y - context.Font.Ascent, context.Cursor.Y - context.Font.Descent,
      this.PageSize.Width - this.Margins.Right - context.Translation.X - context.MarginBottomRight.X),
      this);
00251             context.CurrentLine = null;
00252
00253             context.Cursor = new Point(0, context.Cursor.Y - context.Font.Descent + SpaceAfterLine *
      context.Font.FontSize);
00254
00255             if (spaceAfter)
00256             {
00257                 context.Cursor = new Point(0, context.Cursor.Y + SpaceAfterParagraph *
      context.Font.FontSize);
00258             }
00259         }
00260
00261         private void RenderFencedCodeBlock(FencedCodeBlock codeBlock, ref MarkdownContext context, ref
      Graphics graphics, NewPageAction newPageAction, bool spaceBefore, bool spaceAfter)
00262         {
00263             string info = codeBlock.Info;
00264
00265             if (string.IsNullOrEmpty(info) || codeBlock.Lines.Count == 0)
00266             {
00267                 RenderCodeBlock(codeBlock, ref context, ref graphics, newPageAction, spaceBefore,
      spaceAfter);
00268                 return;
00269             }
00270
00271             StringBuilder code = new StringBuilder();
00272
00273             foreach (StringLine line in codeBlock.Lines)
00274             {
00275                 code.Append(line.ToString());
00276                 code.Append('\n');
00277             }
00278
00279             List<List<FormattedString» lines = this.SyntaxHighlighter(code.ToString(0, code.Length -
      1), info);
00280
00281             if (lines == null)
00282             {
00283                 RenderCodeBlock(codeBlock, ref context, ref graphics, newPageAction, spaceBefore,
      spaceAfter);
00284                 return;
00285             }
00286
00287             MarkdownContext prevContext = context.Clone();
00288
00289             context.Font = new Font(this.CodeFont, context.Font.FontSize);
00290
00291             if (spaceBefore)
00292             {
00293                 context.Cursor = new Point(0, context.Cursor.Y + SpaceBeforeParagaph *
      context.Font.FontSize);
00294             }
```

```
00295
00296             int index = 0;
00297
00298             if (codeBlock.Lines.Count > 0)
00299             {
00300                 foreach (List<FormattedString> line in lines)
00301                 {
00302                     if (index < codeBlock.Lines.Count)
00303                     {
00304                         context.CurrentLine ??= new Line(context.Font.Ascent);
00305
00306                         double maxX = context.GetMaxX(context.Cursor.Y - context.Font.Ascent,
      context.Cursor.Y - context.Font.Descent, this.PageSize.Width - this.Margins.Right -
      context.Translation.X - context.MarginBottomRight.X);
00307
00308                         double minX = context.GetMinX(context.Cursor.Y - context.Font.Ascent,
      context.Cursor.Y - context.Font.Descent);
00309
00310                         context.Cursor = new Point(minX + context.Font.FontSize, context.Cursor.Y +
      context.Font.YMax);
00311
00312                         if (index == 0)
00313                         {
00314                             context.CurrentLine.Fragments.Insert(0, new RectangleFragment(new
      Point(minX, context.Cursor.Y - context.Font.YMax - this.SpaceAfterLine * context.Font.FontSize), new
      Size(maxX - minX, this.SpaceAfterLine * context.Font.FontSize * 2), CodeBlockBackgroundColour,
      context.Tag));
00315                         }
00316
00317                         foreach (FormattedString item in line)
00318                         {
00319                             context.Colour = item.Colour;
00320
00321                             if (!item.IsBold && !item.IsItalic)
00322                             {
00323                                 context.Font = new Font(this.CodeFont, context.Font.FontSize);
00324                             }
00325                             else if (item.IsBold && !item.IsItalic)
00326                             {
00327                                 context.Font = new Font(this.CodeFontBold, context.Font.FontSize);
00328                             }
00329                             else if (item.IsBold && item.IsItalic)
00330                             {
00331                                 context.Font = new Font(this.CodeFontBoldItalic,
      context.Font.FontSize);
00332                             }
00333                             else if (!item.IsBold && item.IsItalic)
00334                             {
00335                                 context.Font = new Font(this.CodeFontItalic, context.Font.FontSize);
00336                             }
00337
00338                             RenderCodeBlockLine(item.Text, ref context, ref graphics, newPageAction);
00339                         }
00340
00341
00342                         context.Colour = Colours.Black;
00343
00344                         context.CurrentLine.Render(ref graphics, ref context, newPageAction,
      this.PageSize.Height - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y,
      context.GetMaxX(context.Cursor.Y - context.Font.Ascent, context.Cursor.Y - context.Font.Descent,
      this.PageSize.Width - this.Margins.Right - context.Translation.X - context.MarginBottomRight.X),
      this);
00345                         context.CurrentLine = null;
00346
00347                         context.Cursor = new Point(0, context.Cursor.Y - context.Font.YMin);
00348                         index++;
00349                     }
00350                     else
00351                     {
00352                         break;
00353                     }
00354                 }
00355             }
00356
00357             context.Cursor = new Point(0, context.Cursor.Y + SpaceAfterLine * context.Font.FontSize);
00358
00359             if (spaceAfter)
00360             {
00361                 context.Cursor = new Point(0, context.Cursor.Y + SpaceAfterParagraph *
      context.Font.FontSize);
00362             }
00363
00364             prevContext.Cursor = context.Cursor;
00365             prevContext.BottomRight = context.BottomRight;
00366             prevContext.CurrentPage = context.CurrentPage;
00367             prevContext.CurrentLine = context.CurrentLine;
00368
```

```
00369             context = prevContext;
00370         }
00371
00372         private void RenderCodeBlock(CodeBlock codeBlock, ref MarkdownContext context, ref Graphics
      graphics, NewPageAction newPageAction, bool spaceBefore, bool spaceAfter)
00373         {
00374             MarkdownContext prevContext = context.Clone();
00375
00376             context.Font = new Font(this.CodeFont, context.Font.FontSize);
00377
00378             if (spaceBefore)
00379             {
00380                 context.Cursor = new Point(0, context.Cursor.Y + SpaceBeforeParagaph *
      context.Font.FontSize);
00381             }
00382
00383             int index = 0;
00384
00385             if (codeBlock.Lines.Count > 0)
00386             {
00387                 foreach (StringLine line in codeBlock.Lines)
00388                 {
00389                     if (index < codeBlock.Lines.Count)
00390                     {
00391                         context.CurrentLine ??= new Line(context.Font.Ascent);
00392
00393                         double maxX = context.GetMaxX(context.Cursor.Y - context.Font.Ascent,
      context.Cursor.Y - context.Font.Descent, this.PageSize.Width - this.Margins.Right -
      context.Translation.X - context.MarginBottomRight.X);
00394
00395                         double minX = context.GetMinX(context.Cursor.Y - context.Font.Ascent,
      context.Cursor.Y - context.Font.Descent);
00396
00397                         context.Cursor = new Point(minX + context.Font.FontSize, context.Cursor.Y +
      context.Font.YMax);
00398
00399                         if (index == 0)
00400                         {
00401                             context.CurrentLine.Fragments.Insert(0, new RectangleFragment(new
      Point(minX, context.Cursor.Y - context.Font.YMax - this.SpaceAfterLine * context.Font.FontSize), new
      Size(maxX - minX, this.SpaceAfterLine * context.Font.FontSize * 2), CodeBlockBackgroundColour,
      context.Tag));
00402                         }
00403
00404
00405                         RenderCodeBlockLine(line.ToString(), ref context, ref graphics,
      newPageAction);
00406
00407
00408                         context.Colour = Colours.Black;
00409
00410                         context.CurrentLine.Render(ref graphics, ref context, newPageAction,
      this.PageSize.Height - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y,
      context.GetMaxX(context.Cursor.Y - context.Font.Ascent, context.Cursor.Y - context.Font.Descent,
      this.PageSize.Width - this.Margins.Right - context.Translation.X - context.MarginBottomRight.X),
      this);
00411                         context.CurrentLine = null;
00412
00413                         context.Cursor = new Point(0, context.Cursor.Y - context.Font.YMin);
00414                         index++;
00415                     }
00416                     else
00417                     {
00418                         break;
00419                     }
00420                 }
00421             }
00422
00423             context.Cursor = new Point(0, context.Cursor.Y + SpaceAfterLine * context.Font.FontSize);
00424
00425             if (spaceAfter)
00426             {
00427                 context.Cursor = new Point(0, context.Cursor.Y + SpaceAfterParagraph *
      context.Font.FontSize);
00428             }
00429
00430             prevContext.Cursor = context.Cursor;
00431             prevContext.BottomRight = context.BottomRight;
00432             prevContext.CurrentPage = context.CurrentPage;
00433             prevContext.CurrentLine = context.CurrentLine;
00434
00435             context = prevContext;
00436         }
00437
00438         private void RenderThematicBreakBlock(ThematicBreakBlock thematicBreak, ref MarkdownContext
      context, ref Graphics graphics, NewPageAction newPageAction, bool spaceBefore, bool spaceAfter)
00439         {
```

```
00440             context.CurrentLine ??= new Line(0);
00441
00442             if (spaceBefore)
00443             {
00444                 context.Cursor = new Point(context.Cursor.X, context.Cursor.Y + SpaceBeforeParagaph *
      context.Font.FontSize + this.ThematicBreakThickness * 0.5);
00445             }
00446             else
00447             {
00448                 context.Cursor = new Point(context.Cursor.X, context.Cursor.Y +
      this.ThematicBreakThickness * 0.5);
00449             }
00450
00451
00452             double maxX = context.GetMaxX(context.Cursor.Y, this.PageSize.Width - this.Margins.Right -
      context.Translation.X - context.MarginBottomRight.X);
00453             double minX = context.GetMinX(context.Cursor.Y);
00454
00455             context.CurrentLine.Fragments.Add(new UnderlineFragment(new Point(minX, context.Cursor.Y),
      new Point(maxX, context.Cursor.Y), this.ThematicBreakLineColour, this.ThematicBreakThickness,
      context.Tag));
00456
00457             context.CurrentLine.Render(ref graphics, ref context, newPageAction, this.PageSize.Height
      - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y,
      context.GetMaxX(context.Cursor.Y - context.Font.Ascent, context.Cursor.Y - context.Font.Descent,
      this.PageSize.Width - this.Margins.Right - context.Translation.X - context.MarginBottomRight.X),
      this);
00458             context.CurrentLine = null;
00459
00460             if (spaceAfter)
00461             {
00462                 context.Cursor = new Point(context.Cursor.X, context.Cursor.Y + SpaceAfterParagraph *
      context.Font.FontSize + this.ThematicBreakThickness * 0.5);
00463             }
00464             else
00465             {
00466                 context.Cursor = new Point(context.Cursor.X, context.Cursor.Y +
      this.ThematicBreakThickness * 0.5);
00467             }
00468         }
00469
00470         private void RenderListBlock(ListBlock list, ref MarkdownContext context, ref Graphics
      graphics, NewPageAction newPageAction)
00471         {
00472             MarkdownContext prevContext = context.Clone();
00473
00474             context.ListDepth++;
00475
00476             double minX = context.GetMinX(context.Cursor.Y + SpaceBeforeParagaph *
      context.Font.FontSize, context.Cursor.Y - context.Font.Descent + context.Font.Ascent +
      SpaceBeforeParagaph * context.Font.FontSize);
00477
00478             if (context.CurrentLine != null)
00479             {
00480                 foreach (LineFragment fragment in context.CurrentLine.Fragments)
00481                 {
00482                     fragment.Translate(-minX - this.IndentWidth, 0);
00483                 }
00484             }
00485
00486             graphics.Translate(minX + this.IndentWidth, 0);
00487             context.Translation = new Point(context.Translation.X + minX + this.IndentWidth,
      context.Translation.Y);
00488
00489             foreach (Block block in list)
00490             {
00491                 RenderBlock(block, ref context, ref graphics, newPageAction, true, true);
00492             }
00493
00494             graphics.Translate(-minX - this.IndentWidth, 0);
00495             context.Translation = new Point(context.Translation.X - minX - this.IndentWidth,
      context.Translation.Y);
00496
00497             prevContext.Cursor = context.Cursor;
00498             prevContext.BottomRight = context.BottomRight;
00499             prevContext.CurrentPage = context.CurrentPage;
00500             prevContext.CurrentLine = context.CurrentLine;
00501
00502             context = prevContext;
00503         }
00504
00505         private void RenderQuoteBlock(QuoteBlock quote, ref MarkdownContext context, ref Graphics
      graphics, NewPageAction newPageAction)
00506         {
00507             MarkdownContext prevContext = context.Clone();
00508
00509             double minX = context.GetMinX(context.Cursor.Y + SpaceBeforeParagaph *
```

```
      context.Font.FontSize, context.Cursor.Y - context.Font.Descent + context.Font.Ascent +
      SpaceBeforeParagaph * context.Font.FontSize);
00510
00511             Graphics quoteGraphics = new Graphics();
00512
00513             quoteGraphics.Translate(minX + this.QuoteBlockIndentWidth, 0);
00514             context.Translation = new Point(minX + this.QuoteBlockIndentWidth, 0);
00515             context.MarginBottomRight = new Point(context.MarginBottomRight.X +
      prevContext.Translation.X, context.MarginBottomRight.Y + prevContext.Translation.Y);
00516
00517             double maxX = this.PageSize.Width - this.Margins.Right - context.Translation.X -
      context.MarginBottomRight.X;
00518
00519             double startY = context.Cursor.Y + context.Font.Ascent - context.Font.YMax;
00520
00521             Point currTranslation = context.Translation;
00522
00523             Graphics parentGraphics = graphics;
00524
00525             void newPageActionWithBlockquotes(ref MarkdownContext currContext, ref Graphics
      currGraphics)
00526             {
00527                 double currEndY = currContext.Cursor.Y;
00528
00529                 double currMaxX = maxX;
00530
00531                 Graphics currBackgroundGraphics = new Graphics();
00532
00533                 currBackgroundGraphics.Save();
00534
00535                 currBackgroundGraphics.Translate(currContext.Translation);
00536                 currBackgroundGraphics.FillRectangle(new Point(-this.QuoteBlockIndentWidth +
      this.QuoteBlockBarWidth, startY), new Size(currMaxX + this.QuoteBlockIndentWidth -
      this.QuoteBlockBarWidth, currEndY - startY), this.QuoteBlockBackgroundColour, tag:  currContext.Tag);
00537                 currBackgroundGraphics.FillRectangle(new Point(-this.QuoteBlockIndentWidth, startY),
      new Size(this.QuoteBlockBarWidth, currEndY - startY), this.QuoteBlockBarColour, tag:
      currContext.Tag);
00538
00539                 currBackgroundGraphics.Restore();
00540
00541                 currBackgroundGraphics.DrawGraphics(0, 0, currGraphics);
00542
00543                 parentGraphics.DrawGraphics(0, 0, currBackgroundGraphics);
00544
00545                 Point currContextTranslation = currContext.Translation;
00546
00547                 currContext.Translation = prevContext.Translation;
00548
00549                 newPageAction(ref currContext, ref parentGraphics);
00550
00551                 currContext.Translation = currContextTranslation;
00552
00553                 currGraphics = new Graphics();
00554                 currGraphics.Translate(minX + this.QuoteBlockIndentWidth, 0);
00555
00556                 startY = currContext.Cursor.Y + currContext.Font.Ascent - currContext.Font.YMax;
00557             }
00558
00559             int index = 0;
00560
00561             foreach (Block block in quote)
00562             {
00563                 RenderBlock(block, ref context, ref quoteGraphics, newPageActionWithBlockquotes, true,
      index < quote.Count - 1);
00564                 index++;
00565             }
00566
00567             double endY = context.Cursor.Y;
00568
00569             Graphics backgroundGraphics = new Graphics();
00570
00571             backgroundGraphics.Save();
00572
00573             backgroundGraphics.Translate(context.Translation);
00574             backgroundGraphics.FillRectangle(new Point(-this.QuoteBlockIndentWidth +
      this.QuoteBlockBarWidth, startY), new Size(maxX + this.QuoteBlockIndentWidth -
      this.QuoteBlockBarWidth, endY - startY), this.QuoteBlockBackgroundColour, tag:  context.Tag);
00575             backgroundGraphics.FillRectangle(new Point(-this.QuoteBlockIndentWidth, startY), new
      Size(this.QuoteBlockBarWidth, endY - startY), this.QuoteBlockBarColour, tag:  context.Tag);
00576
00577             backgroundGraphics.Restore();
00578
00579             backgroundGraphics.DrawGraphics(0, 0, quoteGraphics);
00580
00581             parentGraphics.DrawGraphics(0, 0, backgroundGraphics);
00582
00583             graphics = parentGraphics;
```

```
00584
00585             context.Translation = prevContext.Translation;
00586
00587             if (!(quote.Parent is QuoteBlock) || quote.Parent.LastChild != quote)
00588             {
00589                 context.Cursor = new Point(context.Cursor.X, context.Cursor.Y + SpaceAfterParagraph *
      context.Font.FontSize);
00590             }
00591
00592             prevContext.Cursor = context.Cursor;
00593             prevContext.BottomRight = context.BottomRight;
00594             prevContext.CurrentPage = context.CurrentPage;
00595             prevContext.CurrentLine = context.CurrentLine;
00596
00597             context = prevContext;
00598         }
00599
00600         private void RenderListItemBlock(ListItemBlock listItem, ref MarkdownContext context, ref
      Graphics graphics, NewPageAction newPageAction)
00601         {
00602             MarkdownContext prevContext = context.Clone();
00603
00604             bool isLoose = false;
00605
00606             double startX = context.Cursor.X;
00607             double startY = context.Cursor.Y;
00608
00609             if (listItem.Parent is ListBlock list)
00610             {
00611                 if (list.IsOrdered)
00612                 {
00613                     context.CurrentLine ??= new Line(context.Font.Ascent);
00614
00615                     string bullet;
00616
00617                     switch (list.BulletType)
00618                     {
00619                         case 'a':
00620                         case 'A':
00621                             bullet = ((char)((int)list.DefaultOrderedStart[0] + listItem.Order -
      1)).ToString() + list.OrderedDelimiter;
00622                             break;
00623                         case 'i':
00624                             bullet = GetRomanNumeral(listItem.Order).ToLower() +
      list.OrderedDelimiter;
00625                             break;
00626                         case 'I':
00627                             bullet = GetRomanNumeral(listItem.Order) + list.OrderedDelimiter;
00628                             break;
00629                         default:
00630                             bullet = listItem.Order.ToString() + list.OrderedDelimiter;
00631                             break;
00632                     }
00633
00634                     if (list.IsLoose)
00635                     {
00636                         isLoose = true;
00637                         context.CurrentLine.Fragments.Add(new TextFragment(new Point(context.Cursor.X
      - context.Font.MeasureText(bullet).Width - this.IndentWidth * 0.15, context.Cursor.Y +
      context.Font.Ascent + SpaceBeforeParagaph * context.Font.FontSize), bullet, context.Font,
      context.Colour, context.Tag));
00638                     }
00639                     else
00640                     {
00641                         isLoose = false;
00642                         context.CurrentLine.Fragments.Add(new TextFragment(new Point(context.Cursor.X
      - context.Font.MeasureText(bullet).Width - this.IndentWidth * 0.15, context.Cursor.Y +
      context.Font.Ascent), bullet, context.Font, context.Colour, context.Tag));
00643                     }
00644                 }
00645                 else
00646                 {
00647                     if (listItem.Count > 0 && listItem[0] is ParagraphBlock paragraph &&
      paragraph.Inline?.FirstChild is Markdig.Extensions.TaskLists.TaskList task)
00648                     {
00649                         context.CurrentLine ??= new Line(context.Font.Ascent);
00650
00651                         Graphics bullet = new Graphics();
00652                         bullet.Scale(context.Font.FontSize, context.Font.FontSize);
00653
00654                         if (task.Checked)
00655                         {
00656                             bullet.DrawGraphics(0, 0, this.TaskListCheckedBullet);
00657                         }
00658                         else
00659                         {
00660                             bullet.DrawGraphics(0, 0, this.TaskListUncheckedBullet);
```

```
00661                       }
00662
00663                       if (list.IsLoose)
00664                       {
00665                           isLoose = true;
00666                           context.CurrentLine.Fragments.Add(new GraphicsFragment(new
      Point(context.Cursor.X - this.IndentWidth * 0.15, context.Cursor.Y + context.Font.Ascent +
      SpaceBeforeParagaph * context.Font.FontSize - context.Font.Descent * 0.5 - (context.Font.Ascent -
      context.Font.Descent) * 0.5), bullet, 0));
00667                       }
00668                       else
00669                       {
00670                           isLoose = false;
00671                           context.CurrentLine.Fragments.Add(new GraphicsFragment(new
      Point(context.Cursor.X - this.IndentWidth * 0.15, context.Cursor.Y + context.Font.Ascent -
      context.Font.Descent * 0.5 - (context.Font.Ascent - context.Font.Descent) * 0.5), bullet, 0));
00672                       }
00673                   }
00674                   else
00675                   {
00676                       context.CurrentLine ??= new Line(context.Font.Ascent);
00677
00678                       Graphics bullet = new Graphics();
00679                       bullet.Scale(context.Font.FontSize, context.Font.FontSize);
00680                       this.Bullets[(context.ListDepth - 1) % this.Bullets.Count](bullet,
      context.Colour);
00681
00682                       if (list.IsLoose)
00683                       {
00684                           isLoose = true;
00685                           context.CurrentLine.Fragments.Add(new GraphicsFragment(new
      Point(context.Cursor.X - this.IndentWidth * 0.15, context.Cursor.Y + context.Font.Ascent +
      SpaceBeforeParagaph * context.Font.FontSize - context.Font.Descent * 0.5 - (context.Font.Ascent -
      context.Font.Descent) * 0.5), bullet, 0));
00686                       }
00687                       else
00688                       {
00689                           isLoose = false;
00690                           context.CurrentLine.Fragments.Add(new GraphicsFragment(new
      Point(context.Cursor.X - this.IndentWidth * 0.15, context.Cursor.Y + context.Font.Ascent -
      context.Font.Descent * 0.5 - (context.Font.Ascent - context.Font.Descent) * 0.5), bullet, 0));
00691                       }
00692                   }
00693               }
00694           }
00695
00696           foreach (Block block in listItem)
00697           {
00698               RenderBlock(block, ref context, ref graphics, newPageAction, isLoose || listItem ==
      listItem.Parent[0], isLoose || listItem == listItem.Parent.LastChild);
00699           }
00700
00701           prevContext.Cursor = context.Cursor;
00702           prevContext.BottomRight = context.BottomRight;
00703           prevContext.CurrentPage = context.CurrentPage;
00704           prevContext.CurrentLine = context.CurrentLine;
00705
00706           context = prevContext;
00707       }
00708
00709       private void RenderHTMLBlock(string html, bool isInline, ref MarkdownContext context, ref
      Graphics graphics, NewPageAction newPageAction, bool spaceBefore, bool spaceAfter)
00710       {
00711           if (!isInline)
00712           {
00713               double minX = context.GetMinX(context.Cursor.Y + SpaceBeforeParagaph *
      context.Font.FontSize, context.Cursor.Y + context.Font.Ascent + SpaceBeforeParagaph *
      context.Font.FontSize - context.Font.Descent);
00714
00715               context.Cursor = new Point(minX, context.Cursor.Y + context.Font.Ascent +
      SpaceBeforeParagaph * context.Font.FontSize);
00716
00717               context.CurrentLine ??= new Line(context.Font.Ascent);
00718           }
00719
00720           foreach (HtmlTag tag in HtmlTag.Parse(html))
00721           {
00722               if (tag.Tag.Equals("img", StringComparison.OrdinalIgnoreCase) ||
      tag.Tag.Equals("image", StringComparison.OrdinalIgnoreCase))
00723               {
00724                   RenderHTMLImage(tag, isInline, ref context, ref graphics, newPageAction);
00725               }
00726               else if (tag.Tag.Equals("br", StringComparison.OrdinalIgnoreCase))
00727               {
00728                   RenderLineBreakInline(true, tag.Attributes.TryGetValue("type", out string
      typeValue) && typeValue.Equals("page", StringComparison.OrdinalIgnoreCase) && this.AllowPageBreak, ref
      context, ref graphics, newPageAction);
```

```
00729                 }
00730                 else if (tag.Tag.Equals("a"))
00731                 {
00732                     if (tag.Attributes.TryGetValue("name", out string anchorName))
00733                     {
00734                         if (!context.InternalAnchors.TryGetValue("#" + anchorName, out string anchor))
00735                         {
00736                             anchor = Guid.NewGuid().ToString("N");
00737                             context.InternalAnchors["#" + anchorName] = anchor;
00738                         }
00739 
00740                         context.CurrentLine ??= new Line(0);
00741 
00742                         double anchorHeight = this.BaseFontSize * Math.Max(1,
      this.HeaderFontSizeMultipliers.Max());
00743 
00744                         context.CurrentLine.Fragments.Add(new RectangleFragment(new
      Point(context.Cursor.X, context.Cursor.Y - anchorHeight), new Size(10, anchorHeight),
      Colour.FromRgba(0, 0, 0, 0), anchor));
00745                     }
00746                 }
00747                 else
00748                 {
00749 
00750                 }
00751             }
00752 
00753             if (!isInline)
00754             {
00755                 if (context.CurrentLine != null)
00756                 {
00757                     context.CurrentLine.Render(ref graphics, ref context, newPageAction,
      this.PageSize.Height - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y,
      context.GetMaxX(context.Cursor.Y - context.Font.Ascent, context.Cursor.Y - context.Font.Descent,
      this.PageSize.Width - this.Margins.Right - context.Translation.X - context.MarginBottomRight.X),
      this);
00758                     context.CurrentLine = null;
00759 
00760                     context.Cursor = new Point(0, context.Cursor.Y - context.Font.Descent +
      SpaceAfterLine * context.Font.FontSize);
00761 
00762                     context.Cursor = new Point(0, context.Cursor.Y + SpaceAfterParagraph *
      context.Font.FontSize);
00763                 }
00764             }
00765         }
00766 
00767     }
00768 }
```

8.12 MarkdownRenderer.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using VectSharp.Markdown.CSharpMath.VectSharp;
00019 using Markdig;
00020 using Markdig.Extensions;
00021 using Markdig.Extensions.Tables;
00022 using Markdig.Helpers;
00023 using Markdig.Renderers.Html;
00024 using Markdig.Syntax;
00025 using Markdig.Syntax.Inlines;
00026 using System;
00027 using System.Collections.Generic;
00028 using System.IO;
00029 using System.Linq;
00030 using System.Text;
00031 using Markdig.Extensions.Emoji;
```

```
00032
00033 namespace VectSharp.Markdown
00034 {
00035 /// <summary>
00036 /// Renders Markdown documents into VectSharp graphics objects.
00037 /// </summary>
00038     public partial class MarkdownRenderer
00039     {
00040
00041 /// <summary>
00042 /// Create a new <see cref="MarkdownRenderer"/>;
00043 /// </summary>
00044         public MarkdownRenderer()
00045         {
00046             this.EmojiUriResolver = x => HTTPUtils.ResolveEmojiUsingOpenMoji(x, this);
00047         }
00048
00049         internal MarkdownRenderer Clone()
00050         {
00051             return (MarkdownRenderer)this.MemberwiseClone();
00052         }
00053
00054         private MathPainter MathPainter = new MathPainter() { DisplayErrorInline = false };
00055
00056         internal delegate void NewPageAction(ref MarkdownContext context, ref Graphics graphics);
00057
00058 /// <summary>
00059 /// Parses the supplied <paramref name="markdownSource"/> using all the supported extensions and
      renders the resulting document.  Page breaks are disabled, and the document is rendered as a single
      page with the specified <paramref name="width"/>.  The page will be cropped at the appropriate height
      to contain the entire document.
00060 /// </summary>
00061 /// <param name="markdownSource">The markdown source to parse.</param>
00062 /// <param name="width">The width of the page.</param>
00063 /// <param name="linkDestinations">When this method returns, this value will contain a dictionary used
      to associate graphic action tags to hyperlinks.  This can be used to enable such links when rendering
      the <see cref="Page"/> to a file.</param>
00064 /// <param name="headingTree">When this method returns, this object will contain a list of the
      headings contained in the document and the corresponding tag.  This can be used to create a document
      outline.</param>
00065 /// <returns>A <see cref="Page"/> containing a rendering of the supplied markdown document.</returns>
00066         public Page RenderSinglePage(string markdownSource, double width, out Dictionary<string,
      string> linkDestinations, out List<(int level, string heading, string tag)> headingTree)
00067         {
00068             MarkdownDocument document = Markdig.Markdown.Parse(markdownSource,
      this.MarkdownPipelineBuilder.Build());
00069
00070             return this.RenderSinglePage(document, width, out linkDestinations, out headingTree);
00071         }
00072
00073 /// <summary>
00074 /// Renders the supplied <paramref name="markdownDocument"/>.  Page breaks are disabled, and the
      document is rendered as a single page with the specified <paramref name="width"/>.  The page will be
      cropped at the appropriate height to contain the entire document.
00075 /// </summary>
00076 /// <param name="markdownDocument">The markdown document to render.</param>
00077 /// <param name="width">The width of the page.</param>
00078 /// <param name="linkDestinations">When this method returns, this value will contain a dictionary used
      to associate graphic action tags to hyperlinks.  This can be used to enable such links when rendering
      the <see cref="Page"/> to a file.</param>
00079 /// <param name="headingTree">When this method returns, this object will contain a list of the
      headings contained in the document and the corresponding tag.  This can be used to create a document
      outline.</param>
00080 /// <returns>A <see cref="Page"/> containing a rendering of the supplied markdown document.</returns>
00081         public Page RenderSinglePage(MarkdownDocument markdownDocument, double width, out
      Dictionary<string, string> linkDestinations, out List<(int level, string heading, string tag)>
      headingTree)
00082         {
00083             Size prevPageSize = this.PageSize;
00084             bool allowPageBreak = this.AllowPageBreak;
00085
00086             this.PageSize = new Size(width, double.PositiveInfinity);
00087             this.AllowPageBreak = false;
00088
00089             Page pag = new Page(PageSize.Width, PageSize.Height) { Background = BackgroundColour };
00090
00091             Graphics graphics = pag.Graphics;
00092
00093             static void newPageAction(ref MarkdownContext mdContext, ref Graphics pageGraphics)
00094             {
00095
00096             }
00097
00098             MarkdownContext context = new MarkdownContext()
00099             {
00100                 Font = new Font(RegularFontFamily, BaseFontSize),
00101                 Cursor = new Point(0, 0),
```

```
00102                 Colour = ForegroundColour,
00103                 Underline = false,
00104                 Translation = new Point(this.Margins.Left, this.Margins.Top),
00105                 CurrentLine = null,
00106                 ListDepth = 0,
00107                 CurrentPage = pag
00108             };
00109 
00110             this.OnPageStarted(ref context, ref graphics, pag);
00111 
00112             graphics.Save();
00113 
00114             graphics.Translate(Margins.Left, Margins.Top);
00115 
00116             int index = 0;
00117             foreach (Block block in markdownDocument)
00118             {
00119                 RenderBlock(block, ref context, ref graphics, newPageAction, index > 0, index <
      markdownDocument.Count - 1);
00120                 index++;
00121             }
00122 
00123             context.CurrentLine?.Render(ref graphics, ref context, newPageAction, this.PageSize.Height
      - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y,
      context.GetMaxX(context.Cursor.Y - context.Font.Ascent, context.Cursor.Y - context.Font.Descent,
      this.PageSize.Width - this.Margins.Right - context.Translation.X - context.MarginBottomRight.X),
      this);
00124 
00125             graphics.Restore();
00126 
00127             pag.Crop(new Point(0, 0), new Size(width, context.BottomRight.Y + this.Margins.Bottom));
00128 
00129             this.OnPageFinished(ref context, ref graphics, context.CurrentPage);
00130 
00131             linkDestinations = context.LinkDestinations;
00132             headingTree = context.Headings;
00133             return pag;
00134         }
00135 
00136 /// <summary>
00137 /// Parses the supplied <paramref name="markdownSource"/> using all the supported extensions and
      renders the resulting document.  The <see cref="Document"/> produced consists of one or more pages of
      the size specified in the <see cref="PageSize"/> of the current instance.
00138 /// </summary>
00139 /// <param name="markdownSource">The markdown source to parse.</param>
00140 /// <param name="linkDestinations">When this method returns, this value will contain a dictionary used
      to associate graphic action tags to hyperlinks.  This can be used to enable such links when rendering
      the <see cref="Document"/> to a file.</param>
00141 /// <param name="headingTree">When this method returns, this object will contain a list of the
      headings contained in the document and the corresponding tag.  This can be used to create a document
      outline.</param>
00142 /// <returns>A <see cref="Document"/> containing a rendering of the supplied markdown document,
      consisting of one or more pages of the size specified in the <see cref="PageSize"/> of the current
      instance.</returns>
00143         public Document Render(string markdownSource, out Dictionary<string, string> linkDestinations,
      out List<(int level, string heading, string tag)> headingTree)
00144         {
00145             MarkdownDocument document = Markdig.Markdown.Parse(markdownSource,
      this.MarkdownPipelineBuilder.Build());
00146 
00147             return this.Render(document, out linkDestinations, out headingTree);
00148         }
00149 
00150 /// <summary>
00151 /// Renders the supplied <paramref name="mardownDocument"/>.  The <see cref="Document"/> produced
      consists of one or more pages of the size specified in the <see cref="PageSize"/> of the current
      instance.
00152 /// </summary>
00153 /// <param name="mardownDocument">The markdown document to render.</param>
00154 /// <param name="linkDestinations">When this method returns, this object will contain a dictionary
      used to associate graphic action tags to hyperlinks.  This can be used to enable such links when
      rendering the <see cref="Document"/> to a file.</param>
00155 /// <param name="headingTree">When this method returns, this object will contain a list of the
      headings contained in the document and the corresponding tag.  This can be used to create a document
      outline.</param>
00156 /// <returns>A <see cref="Document"/> containing a rendering of the supplied markdown document,
      consisting of one or more pages of the size specified in the <see cref="PageSize"/> of the current
      instance.</returns>
00157         public Document Render(MarkdownDocument mardownDocument, out Dictionary<string, string>
      linkDestinations, out List<(int level, string heading, string tag)> headingTree)
00158         {
00159             Document doc = new Document();
00160             Page pag = new Page(PageSize.Width, PageSize.Height) { Background = BackgroundColour };
00161             doc.Pages.Add(pag);
00162 
00163             Graphics graphics = pag.Graphics;
00164 
```

```
00165             void newPageAction(ref MarkdownContext mdContext, ref Graphics pageGraphics)
00166             {
00167                 mdContext.CurrentPage.Graphics.Translate(-mdContext.Translation.X,
      -mdContext.Translation.Y);
00168                 this.OnPageFinished(ref mdContext, ref pageGraphics, mdContext.CurrentPage);
00169 
00170                 Page newPag = new Page(PageSize.Width, PageSize.Height) { Background =
      BackgroundColour };
00171                 doc.Pages.Add(newPag);
00172 
00173                 mdContext.StartNewPage(newPag);
00174                 pageGraphics = newPag.Graphics;
00175 
00176                 this.OnPageStarted(ref mdContext, ref pageGraphics, newPag);
00177 
00178                 newPag.Graphics.Translate(mdContext.Translation);
00179             }
00180 
00181             MarkdownContext context = new MarkdownContext()
00182             {
00183                 Font = new Font(RegularFontFamily, BaseFontSize),
00184                 Cursor = new Point(0, 0),
00185                 Colour = ForegroundColour,
00186                 Underline = false,
00187                 Translation = new Point(Margins.Left, Margins.Top),
00188                 CurrentLine = null,
00189                 ListDepth = 0,
00190                 CurrentPage = pag
00191             };
00192 
00193             this.OnPageStarted(ref context, ref graphics, pag);
00194 
00195             graphics.Translate(context.Translation.X, context.Translation.Y);
00196 
00197             int index = 0;
00198 
00199             foreach (Block block in mardownDocument)
00200             {
00201                 RenderBlock(block, ref context, ref graphics, newPageAction, index > 0, index <
      mardownDocument.Count - 1);
00202                 index++;
00203             }
00204 
00205             context.CurrentLine?.Render(ref graphics, ref context, newPageAction, this.PageSize.Height
      - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y,
      context.GetMaxX(context.Cursor.Y - context.Font.Ascent, context.Cursor.Y - context.Font.Descent,
      this.PageSize.Width - this.Margins.Right - context.Translation.X - context.MarginBottomRight.X),
      this);
00206 
00207             context.CurrentPage.Graphics.Translate(-context.Translation.X, -context.Translation.Y);
00208             this.OnPageFinished(ref context, ref graphics, context.CurrentPage);
00209 
00210             linkDestinations = context.LinkDestinations;
00211             headingTree = context.Headings;
00212 
00213             return doc;
00214         }
00215 
00216         private void RenderHTMLImage(HtmlTag imgTag, bool isInline, ref MarkdownContext context, ref
      Graphics graphics, NewPageAction newPageAction)
00217         {
00218             if (imgTag != null)
00219             {
00220                 if (imgTag.Attributes.TryGetValue("src", out string imageSrc))
00221                 {
00222                     string imageFile;
00223                     bool wasDownloaded;
00224 
00225                     if (imageSrc.StartsWith("emoji://") || imageSrc.StartsWith("unicode://"))
00226                     {
00227                         (imageFile, wasDownloaded) = this.EmojiUriResolver(imageSrc);
00228                     }
00229                     else
00230                     {
00231                         (imageFile, wasDownloaded) = this.ImageUriResolver(imageSrc,
      this.BaseImageUri);
00232                     }
00233 
00234                     Page imagePage = null;
00235 
00236                     if (imageFile != null)
00237                     {
00238                         if (System.IO.Path.GetExtension(imageFile) == ".svg")
00239                         {
00240                             try
00241                             {
00242                                 imagePage = VectSharp.SVG.Parser.FromFile(imageFile);
```

```
00243                           }
00244                           catch
00245                           {
00246                               imagePage = null;
00247                           }
00248                       }
00249                       else if (RasterImageLoader != null)
00250                       {
00251                           try
00252                           {
00253                               RasterImage raster = RasterImageLoader(imageFile);
00254                               imagePage = new Page(raster.Width, raster.Height);
00255                               imagePage.Graphics.DrawRasterImage(0, 0, raster, tag:  context.Tag);
00256                           }
00257                           catch
00258                           {
00259                               imagePage = null;
00260                           }
00261                       }
00262 
00263 
00264                       if (wasDownloaded)
00265                       {
00266                           System.IO.File.Delete(imageFile);
00267                           System.IO.Directory.Delete(System.IO.Path.GetDirectoryName(imageFile));
00268                       }
00269                   }
00270                   else if (imageSrc.StartsWith("data:"))
00271                   {
00272                       try
00273                       {
00274                           imagePage = VectSharp.SVG.Parser.ParseImageURI(imageSrc, false);
00275                       }
00276                       catch
00277                       {
00278 
00279                           imagePage = null;
00280                       }
00281                   }
00282 
00283                   if (imagePage != null)
00284                   {
00285                       double scaleX = 1;
00286                       double scaleY = 1;
00287 
00288                       bool hasWidth = false;
00289                       bool hasHeight = false;
00290 
00291                       if (imgTag.Attributes.TryGetValue("width", out string widthString) &&
      double.TryParse(widthString, System.Globalization.NumberStyles.Any,
      System.Globalization.CultureInfo.InvariantCulture, out double width))
00292                       {
00293                           hasWidth = true;
00294                           scaleX = width * this.ImageUnitMultiplier / imagePage.Width;
00295                       }
00296 
00297                       if (imgTag.Attributes.TryGetValue("height", out string heightString) &&
      double.TryParse(heightString, System.Globalization.NumberStyles.Any,
      System.Globalization.CultureInfo.InvariantCulture, out double height))
00298                       {
00299                           hasHeight = true;
00300                           scaleY = height * this.ImageUnitMultiplier / imagePage.Height;
00301                       }
00302 
00303                       if (hasWidth && !hasHeight)
00304                       {
00305                           scaleY = scaleX;
00306                       }
00307                       else if (hasHeight && !hasWidth)
00308                       {
00309                           scaleX = scaleY;
00310                       }
00311 
00312                       scaleX *= this.ImageMultiplier;
00313                       scaleY *= this.ImageMultiplier;
00314 
00315                       if (!isInline)
00316                       {
00317                           if (!imgTag.Attributes.TryGetValue("align", out string alignValue))
00318                           {
00319                               alignValue = null;
00320                           }
00321 
00322                           if (alignValue == "center")
00323                           {
00324                               if (context.CurrentLine != null)
00325                               {
```

```
00326                                   context.CurrentLine.Render(ref graphics, ref context,
      newPageAction, this.PageSize.Height - this.Margins.Bottom - context.Translation.Y -
      context.MarginBottomRight.Y, context.GetMaxX(context.Cursor.Y - context.Font.Ascent, context.Cursor.Y
      - context.Font.Descent, this.PageSize.Width - this.Margins.Right - context.Translation.X -
      context.MarginBottomRight.X), this);
00327                                   context.CurrentLine = null;
00328                               }
00329
00330                               double minX = context.GetMinX(context.Cursor.Y +
      context.Translation.Y, context.Cursor.Y + context.Translation.Y + scaleY * imagePage.Height);
00331                               double maxX = context.GetMaxX(context.Cursor.Y +
      context.Translation.Y, context.Cursor.Y + context.Translation.Y + scaleY * imagePage.Height,
      this.PageSize.Width - this.Margins.Right - context.Translation.X - context.MarginBottomRight.X);
00332
00333                               if (scaleX * imagePage.Width > maxX - minX)
00334                               {
00335                                   scaleX = (maxX - minX) / imagePage.Width;
00336                                   scaleY = scaleX;
00337                               }
00338
00339                               double finalY = context.Cursor.Y + scaleY * imagePage.Height;
00340
00341                               if (finalY + context.Translation.Y > this.PageSize.Height -
      this.Margins.Bottom + this.ImageMarginTolerance - context.MarginBottomRight.Y)
00342                               {
00343                                   newPageAction(ref context, ref graphics);
00344                                   finalY = context.Cursor.Y + scaleY * imagePage.Height;
00345                               }
00346
00347                               graphics.Save();
00348                               graphics.Translate((minX + maxX - scaleX * imagePage.Width) * 0.5,
      context.Cursor.Y);
00349                               graphics.Scale(scaleX, scaleY);
00350                               graphics.SetClippingPath(0, 0, imagePage.Width, imagePage.Height);
00351                               graphics.DrawGraphics(0, 0, imagePage.Graphics);
00352
00353                               graphics.Restore();
00354
00355                               context.Cursor = new Point(0, finalY + SpaceAfterParagraph *
      context.Font.FontSize + SpaceAfterLine * context.Font.FontSize);
00356                           }
00357                           else if (alignValue == "right")
00358                           {
00359                               if (context.CurrentLine != null)
00360                               {
00361                                   context.CurrentLine.Render(ref graphics, ref context,
      newPageAction, this.PageSize.Height - this.Margins.Bottom - context.Translation.Y -
      context.MarginBottomRight.Y, context.GetMaxX(context.Cursor.Y - context.Font.Ascent, context.Cursor.Y
      - context.Font.Descent, this.PageSize.Width - this.Margins.Right - context.Translation.X -
      context.MarginBottomRight.X), this);
00362                                   context.CurrentLine = null;
00363                               }
00364
00365                               double finalY = context.Cursor.Y + scaleY * imagePage.Height;
00366
00367                               if (finalY + context.Translation.Y > this.PageSize.Height -
      this.Margins.Bottom + this.ImageMarginTolerance - context.MarginBottomRight.Y)
00368                               {
00369                                   newPageAction(ref context, ref graphics);
00370                               }
00371
00372                               graphics.Save();
00373                               graphics.Translate(this.PageSize.Width - this.Margins.Right -
      context.Translation.X - scaleX * imagePage.Width - context.MarginBottomRight.X, context.Cursor.Y);
00374                               graphics.Scale(scaleX, scaleY);
00375                               graphics.SetClippingPath(0, 0, imagePage.Width, imagePage.Height);
00376
00377                               graphics.DrawGraphics(0, 0, imagePage.Graphics);
00378
00379                               graphics.Restore();
00380
00381                               context.ForbiddenAreasRight.Add((this.PageSize.Width -
      this.Margins.Right - scaleX * imagePage.Width - this.ImageSideMargin - context.MarginBottomRight.X,
      context.Cursor.Y + context.Translation.Y, context.Cursor.Y + context.Translation.Y + scaleY *
      imagePage.Height));
00382                           }
00383                           else if (alignValue == "left")
00384                           {
00385                               if (context.CurrentLine != null)
00386                               {
00387                                   context.CurrentLine.Render(ref graphics, ref context,
      newPageAction, this.PageSize.Height - this.Margins.Bottom - context.Translation.Y -
      context.MarginBottomRight.Y, context.GetMaxX(context.Cursor.Y - context.Font.Ascent, context.Cursor.Y
      - context.Font.Descent, this.PageSize.Width - this.Margins.Right - context.Translation.X -
      context.MarginBottomRight.X), this);
00388                                   context.CurrentLine = null;
00389                               }
```

```
00390
00391                                 double finalY = context.Cursor.Y + scaleY * imagePage.Height;
00392
00393                                 if (finalY + context.Translation.Y > this.PageSize.Height -
    this.Margins.Bottom + this.ImageMarginTolerance - context.MarginBottomRight.Y)
00394                                 {
00395                                     newPageAction(ref context, ref graphics);
00396                                 }
00397
00398                                 graphics.Save();
00399                                 graphics.Translate(0, context.Cursor.Y);
00400                                 graphics.Scale(scaleX, scaleY);
00401                                 graphics.SetClippingPath(0, 0, imagePage.Width, imagePage.Height);
00402
00403                                 graphics.DrawGraphics(0, 0, imagePage.Graphics);
00404
00405                                 graphics.Restore();
00406
00407                                 context.ForbiddenAreasLeft.Add((scaleX * imagePage.Width +
    context.Translation.X + this.ImageSideMargin, context.Cursor.Y + context.Translation.Y,
    context.Cursor.Y + context.Translation.Y + scaleY * imagePage.Height));
00408                             }
00409                             else
00410                             {
00411                                 isInline = true;
00412                             }
00413                         }
00414
00415                         if (isInline)
00416                         {
00417                             Graphics scaledImage = new Graphics();
00418                             scaledImage.Scale(scaleX, scaleY);
00419                             scaledImage.DrawGraphics(0, 0, imagePage.Graphics);
00420
00421                             if (context.CurrentLine == null)
00422                             {
00423                                 context.CurrentLine = new Line(context.Font.Ascent);
00424
00425                                 context.Cursor = new Point(0, context.Cursor.Y);
00426                                 double minX = context.GetMinX(context.Cursor.Y - scaleY *
    imagePage.Height, context.Cursor.Y);
00427                                 context.Cursor = new Point(minX, context.Cursor.Y);
00428                             }
00429
00430                             double currLineMaxX = context.GetMaxX(context.Cursor.Y - scaleY *
    imagePage.Height, context.Cursor.Y, this.PageSize.Width - this.Margins.Right - context.Translation.X -
    context.MarginBottomRight.X);
00431
00432                             double finalX = context.Cursor.X + imagePage.Width;
00433
00434                             if (finalX > currLineMaxX)
00435                             {
00436                                 context.CurrentLine.Render(ref graphics, ref context, newPageAction,
    this.PageSize.Height - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y,
    context.GetMaxX(context.Cursor.Y - context.Font.Ascent, context.Cursor.Y - context.Font.Descent,
    this.PageSize.Width - this.Margins.Right - context.Translation.X - context.MarginBottomRight.X),
    this);
00437
00438                                 context.CurrentLine = new Line(context.Font.Ascent);
00439                                 context.Cursor = new Point(0, context.Cursor.Y - context.Font.Descent
    + SpaceAfterLine * context.Font.FontSize + context.Font.Ascent);
00440
00441                                 double minX = context.GetMinX(context.Cursor.Y - scaleY *
    imagePage.Height * scaleY, context.Cursor.Y);
00442
00443                                 context.Cursor = new Point(minX, context.Cursor.Y);
00444                             }
00445
00446                             context.CurrentLine.Fragments.Add(new GraphicsFragment(new
    Point(context.Cursor.X, context.Cursor.Y - scaleY * imagePage.Height), scaledImage, scaleY *
    imagePage.Height));
00447                             context.Cursor = new Point(context.Cursor.X + scaleX * imagePage.Width,
    context.Cursor.Y);
00448                         }
00449                     }
00450                 }
00451             }
00452         }
00453
00454         static (int, string)[] RomanNumbers = new (int, string)[] { (1000, "M"), (900, "CM"), (500,
    "D"), (400, "CD"), (100, "C"), (90, "XC"), (50, "L"), (40, "XL"), (10, "X"), (9, "IX"), (5, "V"), (4,
    "IV"), (1, "I") };
00455
00456         private static string GetRomanNumeral(int number)
00457         {
00458             StringBuilder tbr = new StringBuilder();
00459
```

```
00460             for (int i = 0; i < RomanNumbers.Length; i++)
00461             {
00462                 while (number >= RomanNumbers[i].Item1)
00463                 {
00464                     tbr.Append(RomanNumbers[i].Item2);
00465                     number -= RomanNumbers[i].Item1;
00466                 }
00467             }
00468 
00469             return tbr.ToString();
00470         }
00471     }
00472 }
```

8.13 MarkdownRenderer.Events.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2024 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using Markdig.Syntax;
00019 using Markdig.Syntax.Inlines;
00020 using System;
00021 
00022 namespace VectSharp.Markdown
00023 {
00024     public partial class MarkdownRenderer
00025     {
00026 /// <summary>
00027 /// Invoked when a new page is started.
00028 /// </summary>
00029         public event EventHandler<PageStartedEventArgs> PageStarted;
00030 
00031 /// <summary>
00032 /// Invoked when a page is finished, before starting a new page (or at the end of the document).
00033 /// </summary>
00034         public event EventHandler<PageFinishedEventArgs> PageFinished;
00035 
00036 /// <summary>
00037 /// Invoked just before rendering a <see cref="Block"/>.
00038 /// </summary>
00039         public event EventHandler<BlockRenderingEventArgs> BlockRendering;
00040 
00041 /// <summary>
00042 /// Invoked just after rendering a <see cref="Block"/>.
00043 /// </summary>
00044         public event EventHandler<BlockRenderedEventArgs> BlockRendered;
00045 
00046 /// <summary>
00047 /// Invoked just before rendering an <see cref="Inline"/>.
00048 /// </summary>
00049         public event EventHandler<InlineRenderingEventArgs> InlineRendering;
00050 
00051 /// <summary>
00052 /// Invoked just after rendering an <see cref="Inline"/>.
00053 /// </summary>
00054         public event EventHandler<InlineRenderedEventArgs> InlineRendered;
00055 
00056 /// <summary>
00057 /// Invoked after content in a <see cref="Line"/> has been measured, but before deciding whether to
      render the line on the current page or on a new page.
00058 /// </summary>
00059         public event EventHandler<LineMeasuredEventArgs> LineMeasured;
00060 
00061 /// <summary>
00062 /// Invoked just before rendering a <see cref="Line"/>, after a new page has been created if
      necessary.
00063 /// </summary>
00064         public event EventHandler<LineEventArgs> LineRendering;
00065 
```

```
00066 /// <summary>
00067 /// Invoked just after rendering a <see cref="Line"/>.
00068 /// </summary>
00069         public event EventHandler<LineEventArgs> LineRendered;
00070 
00071 /// <summary>
00072 /// Raises the <see cref="PageStarted"/> event.
00073 /// </summary>
00074 /// <param name="context">The current <see cref="MarkdownContext"/>.</param>
00075 /// <param name="pageGraphics">The current <see cref="Graphics"/> surface.</param>
00076 /// <param name="pag">The new empty <see cref="Page"/>.</param>
00077         protected virtual void OnPageStarted(ref MarkdownContext context, ref Graphics pageGraphics,
      Page pag)
00078         {
00079             PageStartedEventArgs eventArgs = new PageStartedEventArgs(context, pag, pageGraphics);
00080             PageStarted?.Invoke(this, eventArgs);
00081             context = eventArgs.MarkdownContext;
00082             pageGraphics = eventArgs.Graphics;
00083         }
00084 
00085 /// <summary>
00086 /// Raises the <see cref="PageFinished"/> event.
00087 /// </summary>
00088 /// <param name="context">The current <see cref="MarkdownContext"/>.</param>
00089 /// <param name="pageGraphics">The current <see cref="Graphics"/> surface.</param>
00090 /// <param name="pag">The new empty <see cref="Page"/>.</param>
00091         protected virtual void OnPageFinished(ref MarkdownContext context, ref Graphics pageGraphics,
      Page pag)
00092         {
00093             PageFinishedEventArgs eventArgs = new PageFinishedEventArgs(context, pag, pageGraphics);
00094             PageFinished?.Invoke(this, eventArgs);
00095             context = eventArgs.MarkdownContext;
00096             pageGraphics = eventArgs.Graphics;
00097         }
00098 
00099 /// <summary>
00100 /// Raises the <see cref="BlockRendering"/> event.
00101 /// </summary>
00102 /// <param name="context">The current <see cref="MarkdownContext"/>.</param>
00103 /// <param name="graphics">The current <see cref="Graphics"/> surface.</param>
00104 /// <param name="block">The <see cref="Block"/> that will be rendered.</param>
00105         protected virtual void OnBlockRendering(ref MarkdownContext context, ref Graphics graphics,
      ref Block block)
00106         {
00107             BlockRenderingEventArgs eventArgs = new BlockRenderingEventArgs(context, graphics, block);
00108             BlockRendering?.Invoke(this, eventArgs);
00109             context = eventArgs.MarkdownContext;
00110             graphics = eventArgs.Graphics;
00111             block = eventArgs.Block;
00112         }
00113 
00114 /// <summary>
00115 /// Raises the <see cref="BlockRendered"/> event.
00116 /// </summary>
00117 /// <param name="context">The current <see cref="MarkdownContext"/>.</param>
00118 /// <param name="graphics">The current <see cref="Graphics"/> surface.</param>
00119 /// <param name="block">The <see cref="Block"/> that has been rendered.</param>
00120         protected virtual void OnBlockRendered(ref MarkdownContext context, ref Graphics graphics,
      Block block)
00121         {
00122             BlockRenderedEventArgs eventArgs = new BlockRenderedEventArgs(context, graphics, block);
00123             BlockRendered?.Invoke(this, eventArgs);
00124             context = eventArgs.MarkdownContext;
00125             graphics = eventArgs.Graphics;
00126         }
00127 
00128 /// <summary>
00129 /// Raises the <see cref="InlineRendering"/> event.
00130 /// </summary>
00131 /// <param name="context">The current <see cref="MarkdownContext"/>.</param>
00132 /// <param name="graphics">The current <see cref="Graphics"/> surface.</param>
00133 /// <param name="inline">The <see cref="Inline"/> that will be rendered.</param>
00134         protected virtual void OnInlineRendering(ref MarkdownContext context, ref Graphics graphics,
      ref Inline inline)
00135         {
00136             InlineRenderingEventArgs eventArgs = new InlineRenderingEventArgs(context, graphics,
      inline);
00137             InlineRendering?.Invoke(this, eventArgs);
00138             context = eventArgs.MarkdownContext;
00139             graphics = eventArgs.Graphics;
00140             inline = eventArgs.Inline;
00141         }
00142 
00143 /// <summary>
00144 /// Raises the <see cref="InlineRendered"/> event.
00145 /// </summary>
00146 /// <param name="context">The current <see cref="MarkdownContext"/>.</param>
```

```
00147 /// <param name="graphics">The current <see cref="Graphics"/> surface.</param>
00148 /// <param name="inline">The <see cref="Inline"/> that has been rendered.</param>
00149         protected virtual void OnInlineRendered(ref MarkdownContext context, ref Graphics graphics,
      Inline inline)
00150         {
00151             InlineRenderedEventArgs eventArgs = new InlineRenderedEventArgs(context, graphics,
      inline);
00152             InlineRendered?.Invoke(this, eventArgs);
00153             context = eventArgs.MarkdownContext;
00154             graphics = eventArgs.Graphics;
00155         }
00156 
00157 /// <summary>
00158 /// Raises the <see cref="LineMeasured"/> event.
00159 /// </summary>
00160 /// <param name="context">The current <see cref="VectSharp.Markdown.MarkdownContext"/>.</param>
00161 /// <param name="graphics">The current <see cref="VectSharp.Graphics"/> surface.</param>
00162 /// <param name="line">The line that will be rendered.</param>
00163 /// <param name="pageMaxX">The maximum space available for the line on the x axis.</param>
00164 /// <param name="pageMaxY">The maximum space available for the line on the y axis.</param>
00165 /// <param name="contentMaxX">The maximum x coordinate of the content in the line.</param>
00166 /// <param name="contentMinY">The minimum y coordinate of the content in the line.</param>
00167 /// <param name="contentMaxY">The maximum y coordinate of the content in the line.</param>
00168 /// <param name="baseline">The vertical coordinate of the baseline.</param>
00169         protected virtual void OnLineMeasured(ref MarkdownContext context, ref Graphics graphics, ref
      Line line, double pageMaxX, double pageMaxY, double contentMaxX, double contentMinY, double
      contentMaxY, double baseline)
00170         {
00171             LineMeasuredEventArgs eventArgs = new LineMeasuredEventArgs(context, graphics, line,
      pageMaxX, pageMaxY, contentMaxX, contentMinY, contentMaxY, baseline);
00172             LineMeasured?.Invoke(this, eventArgs);
00173             context = eventArgs.MarkdownContext;
00174             graphics = eventArgs.Graphics;
00175             line = eventArgs.Line;
00176         }
00177 
00178 /// <summary>
00179 /// Raises the <see cref="LineRendering"/> event.
00180 /// </summary>
00181 /// <param name="context">The current <see cref="VectSharp.Markdown.MarkdownContext"/>.</param>
00182 /// <param name="graphics">The current <see cref="VectSharp.Graphics"/> surface.</param>
00183 /// <param name="line">The line that will be rendered.</param>
00184 /// <param name="pageMaxX">The maximum space available for the line on the x axis.</param>
00185 /// <param name="pageMaxY">The maximum space available for the line on the y axis.</param>
00186 /// <param name="contentMaxX">The maximum x coordinate of the content in the line.</param>
00187 /// <param name="contentMinY">The minimum y coordinate of the content in the line.</param>
00188 /// <param name="contentMaxY">The maximum y coordinate of the content in the line.</param>
00189 /// <param name="baseline">The vertical coordinate of the baseline.</param>
00190         protected virtual void OnLineRendering(ref MarkdownContext context, ref Graphics graphics,
      Line line, double pageMaxX, double pageMaxY, double contentMaxX, double contentMinY, double
      contentMaxY, double baseline)
00191         {
00192             LineEventArgs eventArgs = new LineEventArgs(context, graphics, line, pageMaxX, pageMaxY,
      contentMaxX, contentMinY, contentMaxY, baseline);
00193             LineRendering?.Invoke(this, eventArgs);
00194             context = eventArgs.MarkdownContext;
00195             graphics = eventArgs.Graphics;
00196         }
00197 
00198 /// <summary>
00199 /// Raises the <see cref="LineRendered"/> event.
00200 /// </summary>
00201 /// <param name="context">The current <see cref="VectSharp.Markdown.MarkdownContext"/>.</param>
00202 /// <param name="graphics">The current <see cref="VectSharp.Graphics"/> surface.</param>
00203 /// <param name="line">The line that will be rendered.</param>
00204 /// <param name="pageMaxX">The maximum space available for the line on the x axis.</param>
00205 /// <param name="pageMaxY">The maximum space available for the line on the y axis.</param>
00206 /// <param name="contentMaxX">The maximum x coordinate of the content in the line.</param>
00207 /// <param name="contentMinY">The minimum y coordinate of the content in the line.</param>
00208 /// <param name="contentMaxY">The maximum y coordinate of the content in the line.</param>
00209 /// <param name="baseline">The vertical coordinate of the baseline.</param>
00210         protected virtual void OnLineRendered(ref MarkdownContext context, ref Graphics graphics, Line
      line, double pageMaxX, double pageMaxY, double contentMaxX, double contentMinY, double contentMaxY,
      double baseline)
00211         {
00212             LineEventArgs eventArgs = new LineEventArgs(context, graphics, line, pageMaxX, pageMaxY,
      contentMaxX, contentMinY, contentMaxY, baseline);
00213             LineRendered?.Invoke(this, eventArgs);
00214             context = eventArgs.MarkdownContext;
00215             graphics = eventArgs.Graphics;
00216         }
00217 
00218         internal void RaiseLineMeasured(ref MarkdownContext context, ref Graphics graphics, ref Line
      line, double pageMaxX, double pageMaxY, double contentMaxX, double contentMinY, double contentMaxY,
      double baseline)
00219         {
00220             OnLineMeasured(ref context, ref graphics, ref line, pageMaxX, pageMaxY, contentMaxX,
```

```
      contentMinY, contentMaxY, baseline);
00221         }
00222
00223         internal void RaiseLineRendering(ref MarkdownContext context, ref Graphics graphics, Line
      line, double pageMaxX, double pageMaxY, double contentMaxX, double contentMinY, double contentMaxY,
      double baseline)
00224         {
00225             OnLineRendering(ref context, ref graphics, line, pageMaxX, pageMaxY, contentMaxX,
      contentMinY, contentMaxY, baseline);
00226         }
00227
00228         internal void RaiseLineRendered(ref MarkdownContext context, ref Graphics graphics, Line line,
      double pageMaxX, double pageMaxY, double contentMaxX, double contentMinY, double contentMaxY, double
      baseline)
00229         {
00230             OnLineRendered(ref context, ref graphics, line, pageMaxX, pageMaxY, contentMaxX,
      contentMinY, contentMaxY, baseline);
00231         }
00232     }
00233
00234 /// <summary>
00235 /// <see cref="EventArgs"/> for the <see cref="MarkdownRenderer.PageStarted"/> event.
00236 /// </summary>
00237     public class PageStartedEventArgs :  EventArgs
00238     {
00239 /// <summary>
00240 /// The current <see cref="VectSharp.Markdown.MarkdownContext"/>.
00241 /// </summary>
00242         public MarkdownContext MarkdownContext { get; set; }
00243
00244 /// <summary>
00245 /// The new empty <see cref="VectSharp.Page"/>.
00246 /// </summary>
00247         public Page Page { get; }
00248
00249 /// <summary>
00250 /// The <see cref="VectSharp.Graphics"/> surface on which the page contents will be drawn.
00251 /// </summary>
00252         public Graphics Graphics { get; set; }
00253
00254 /// <summary>
00255 /// Create a new <see cref="PageStartedEventArgs"/> instance.
00256 /// </summary>
00257 /// <param name="markdownContext">The current <see
      cref="VectSharp.Markdown.MarkdownContext"/>.</param>
00258 /// <param name="page">The new empty <see cref="VectSharp.Page"/>.</param>
00259 /// <param name="graphics">The <see cref="VectSharp.Graphics"/> surface on which the page contents
      will be drawn.</param>
00260         public PageStartedEventArgs(MarkdownContext markdownContext, Page page, Graphics graphics)
00261         {
00262             this.MarkdownContext = markdownContext;
00263             this.Page = page;
00264             this.Graphics = graphics;
00265         }
00266     }
00267
00268 /// <summary>
00269 /// <see cref="EventArgs"/> for the <see cref="MarkdownRenderer.PageFinished"/> event.
00270 /// </summary>
00271     public class PageFinishedEventArgs :  EventArgs
00272     {
00273 /// <summary>
00274 /// The current <see cref="VectSharp.Markdown.MarkdownContext"/>.
00275 /// </summary>
00276         public MarkdownContext MarkdownContext { get; set; }
00277
00278 /// <summary>
00279 /// The finished <see cref="VectSharp.Page"/>.
00280 /// </summary>
00281         public Page Page { get; }
00282
00283 /// <summary>
00284 /// The <see cref="VectSharp.Graphics"/> surface on which the page contents have been drawn.
00285 /// </summary>
00286         public Graphics Graphics { get; set; }
00287
00288 /// <summary>
00289 /// Create a new <see cref="PageFinishedEventArgs"/> instance.
00290 /// </summary>
00291 /// <param name="markdownContext">The current <see
      cref="VectSharp.Markdown.MarkdownContext"/>.</param>
00292 /// <param name="page">The finished <see cref="VectSharp.Page"/>.</param>
00293 /// <param name="graphics">The <see cref="VectSharp.Graphics"/> surface on which the page contents
      have been drawn.</param>
00294         public PageFinishedEventArgs(MarkdownContext markdownContext, Page page, Graphics graphics)
00295         {
00296             this.MarkdownContext = markdownContext;
```

```
00297             this.Page = page;
00298             this.Graphics = graphics;
00299         }
00300     }
00301 
00302 /// <summary>
00303 /// <see cref="EventArgs"/> for the <see cref="MarkdownRenderer.BlockRendering"/> event.
00304 /// </summary>
00305     public class BlockRenderingEventArgs :  EventArgs
00306     {
00307 /// <summary>
00308 /// The current <see cref="VectSharp.Markdown.MarkdownContext"/>.
00309 /// </summary>
00310         public MarkdownContext MarkdownContext { get; set; }
00311 
00312 /// <summary>
00313 /// The <see cref="VectSharp.Graphics"/> surface on which the <see cref="Block"/> will be drawn.
00314 /// </summary>
00315         public Graphics Graphics { get; set; }
00316 
00317 /// <summary>
00318 /// The <see cref="Markdig.Syntax.Block"/> that will be rendered.
00319 /// </summary>
00320         public Block Block { get; set; }
00321 
00322 /// <summary>
00323 /// Create a new <see cref="BlockRenderingEventArgs"/> instance.
00324 /// </summary>
00325 /// <param name="markdownContext">The current <see
      cref="VectSharp.Markdown.MarkdownContext"/>.</param>
00326 /// <param name="graphics">The <see cref="VectSharp.Graphics"/> surface on which the <see
      cref="Block"/> will be drawn.</param>
00327 /// <param name="block">The <see cref="Markdig.Syntax.Block"/> that will be rendered.</param>
00328         public BlockRenderingEventArgs(MarkdownContext markdownContext, Graphics graphics, Block
      block)
00329         {
00330             MarkdownContext = markdownContext;
00331             Graphics = graphics;
00332             Block = block;
00333         }
00334     }
00335 
00336 /// <summary>
00337 /// <see cref="EventArgs"/> for the <see cref="MarkdownRenderer.BlockRendered"/> event.
00338 /// </summary>
00339     public class BlockRenderedEventArgs :  EventArgs
00340     {
00341 /// <summary>
00342 /// The current <see cref="VectSharp.Markdown.MarkdownContext"/>.
00343 /// </summary>
00344         public MarkdownContext MarkdownContext { get; set; }
00345 
00346 /// <summary>
00347 /// The current <see cref="VectSharp.Graphics"/> surface.
00348 /// </summary>
00349         public Graphics Graphics { get; set; }
00350 
00351 /// <summary>
00352 /// The <see cref="Markdig.Syntax.Block"/> that has been rendered.
00353 /// </summary>
00354         public Block Block { get; }
00355 
00356 /// <summary>
00357 /// Create a new <see cref="BlockRenderingEventArgs"/> instance.
00358 /// </summary>
00359 /// <param name="markdownContext">The current <see
      cref="VectSharp.Markdown.MarkdownContext"/>.</param>
00360 /// <param name="graphics">The current <see cref="VectSharp.Graphics"/> surface.</param>
00361 /// <param name="block">The <see cref="Markdig.Syntax.Block"/> that has been rendered.</param>
00362         public BlockRenderedEventArgs(MarkdownContext markdownContext, Graphics graphics, Block block)
00363         {
00364             MarkdownContext = markdownContext;
00365             Graphics = graphics;
00366             Block = block;
00367         }
00368     }
00369 
00370 /// <summary>
00371 /// <see cref="EventArgs"/> for the <see cref="MarkdownRenderer.InlineRendering"/> event.
00372 /// </summary>
00373     public class InlineRenderingEventArgs :  EventArgs
00374     {
00375 /// <summary>
00376 /// The current <see cref="VectSharp.Markdown.MarkdownContext"/>.
00377 /// </summary>
00378         public MarkdownContext MarkdownContext { get; set; }
00379 
```

```
00380 /// <summary>
00381 /// The <see cref="VectSharp.Graphics"/> surface on which the <see cref="Inline"/> will be drawn.
00382 /// </summary>
00383         public Graphics Graphics { get; set; }
00384
00385 /// <summary>
00386 /// The <see cref="Markdig.Syntax.Inlines.Inline"/> that will be rendered.
00387 /// </summary>
00388         public Inline Inline { get; set; }
00389
00390 /// <summary>
00391 /// Create a new <see cref="InlineRenderingEventArgs"/> instance.
00392 /// </summary>
00393 /// <param name="markdownContext">The current <see
      cref="VectSharp.Markdown.MarkdownContext"/>.</param>
00394 /// <param name="graphics">The <see cref="VectSharp.Graphics"/> surface on which the <see
      cref="Inline"/> will be drawn.</param>
00395 /// <param name="inline">The <see cref="Markdig.Syntax.Inlines.Inline"/> that will be
      rendered.</param>
00396         public InlineRenderingEventArgs(MarkdownContext markdownContext, Graphics graphics, Inline
      inline)
00397         {
00398             MarkdownContext = markdownContext;
00399             Graphics = graphics;
00400             Inline = inline;
00401         }
00402     }
00403
00404 /// <summary>
00405 /// <see cref="EventArgs"/> for the <see cref="MarkdownRenderer.InlineRendered"/> event.
00406 /// </summary>
00407     public class InlineRenderedEventArgs :  EventArgs
00408     {
00409 /// <summary>
00410 /// The current <see cref="VectSharp.Markdown.MarkdownContext"/>.
00411 /// </summary>
00412         public MarkdownContext MarkdownContext { get; set; }
00413
00414 /// <summary>
00415 /// The current <see cref="VectSharp.Graphics"/> surface.
00416 /// </summary>
00417         public Graphics Graphics { get; set; }
00418
00419 /// <summary>
00420 /// The <see cref="Markdig.Syntax.Inlines.Inline"/> that has been rendered.
00421 /// </summary>
00422         public Inline Inline { get; }
00423
00424 /// <summary>
00425 /// Create a new <see cref="InlineRenderingEventArgs"/> instance.
00426 /// </summary>
00427 /// <param name="markdownContext">The current <see
      cref="VectSharp.Markdown.MarkdownContext"/>.</param>
00428 /// <param name="graphics">The current <see cref="VectSharp.Graphics"/> surface.</param>
00429 /// <param name="inline">The <see cref="Markdig.Syntax.Inlines.Inline"/> that has been
      rendered.</param>
00430         public InlineRenderedEventArgs(MarkdownContext markdownContext, Graphics graphics, Inline
      inline)
00431         {
00432             MarkdownContext = markdownContext;
00433             Graphics = graphics;
00434             Inline = inline;
00435         }
00436     }
00437
00438 /// <summary>
00439 /// <see cref="EventArgs"/> for the <see cref="MarkdownRenderer.LineMeasured"/> event.
00440 /// </summary>
00441     public class LineMeasuredEventArgs :  EventArgs
00442     {
00443 /// <summary>
00444 /// The current <see cref="VectSharp.Markdown.MarkdownContext"/>.
00445 /// </summary>
00446         public MarkdownContext MarkdownContext { get; set; }
00447
00448 /// <summary>
00449 /// The current <see cref="VectSharp.Graphics"/> surface.
00450 /// </summary>
00451         public Graphics Graphics { get; set; }
00452
00453 /// <summary>
00454 /// The line that will be rendered.
00455 /// </summary>
00456         public Line Line { get; set; }
00457
00458 /// <summary>
00459 /// The maximum space available for the line on the x axis.
```

```
00460 /// </summary>
00461         public double PageMaxX { get; }
00462
00463 /// <summary>
00464 /// The maximum space available for the line on the y axis.
00465 /// </summary>
00466         public double PageMaxY { get; }
00467
00468 /// <summary>
00469 /// The maximum x coordinate of the content in the line.
00470 /// </summary>
00471         public double ContentMaxX { get; }
00472
00473 /// <summary>
00474 /// The minimum y coordinate of the content in the line.
00475 /// </summary>
00476         public double ContentMinY { get; }
00477
00478 /// <summary>
00479 /// The maximum y coordinate of the content in the line.
00480 /// </summary>
00481         public double ContentMaxY { get; }
00482
00483 /// <summary>
00484 /// The vertical coordinate of the baseline.
00485 /// </summary>
00486         public double Baseline { get; }
00487
00488 /// <summary>
00489 /// Create a new <see cref="LineMeasuredEventArgs"/>
00490 /// </summary>
00491 /// <param name="markdownContext">The current <see
      cref="VectSharp.Markdown.MarkdownContext"/>.</param>
00492 /// <param name="graphics">The current <see cref="VectSharp.Graphics"/> surface.</param>
00493 /// <param name="line">The line that will be rendered.</param>
00494 /// <param name="pageMaxX">The maximum space available for the line on the x axis.</param>
00495 /// <param name="pageMaxY">The maximum space available for the line on the y axis.</param>
00496 /// <param name="contentMaxX">The maximum x coordinate of the content in the line.</param>
00497 /// <param name="contentMinY">The minimum y coordinate of the content in the line.</param>
00498 /// <param name="contentMaxY">The maximum y coordinate of the content in the line.</param>
00499 /// <param name="baseline">The vertical coordinate of the baseline.</param>
00500         public LineMeasuredEventArgs(MarkdownContext markdownContext, Graphics graphics, Line line,
      double pageMaxX, double pageMaxY, double contentMaxX, double contentMinY, double contentMaxY, double
      baseline)
00501         {
00502             MarkdownContext = markdownContext;
00503             Graphics = graphics;
00504             Line = line;
00505             PageMaxX = pageMaxX;
00506             PageMaxY = pageMaxY;
00507             ContentMaxX = contentMaxX;
00508             ContentMinY = contentMinY;
00509             ContentMaxY = contentMaxY;
00510             Baseline = baseline;
00511         }
00512     }
00513
00514 /// <summary>
00515 /// <see cref="EventArgs"/> for the <see cref="MarkdownRenderer.LineRendering"/> and <see
      cref="MarkdownRenderer.LineRendered"/> events.
00516 /// </summary>
00517     public class LineEventArgs :  EventArgs
00518     {
00519 /// <summary>
00520 /// The current <see cref="VectSharp.Markdown.MarkdownContext"/>.
00521 /// </summary>
00522         public MarkdownContext MarkdownContext { get; set; }
00523
00524 /// <summary>
00525 /// The current <see cref="VectSharp.Graphics"/> surface.
00526 /// </summary>
00527         public Graphics Graphics { get; set; }
00528
00529 /// <summary>
00530 /// The line that will be rendered.
00531 /// </summary>
00532         public Line Line { get; }
00533
00534 /// <summary>
00535 /// The maximum space available for the line on the x axis.
00536 /// </summary>
00537         public double PageMaxX { get; }
00538
00539 /// <summary>
00540 /// The maximum space available for the line on the y axis.
00541 /// </summary>
00542         public double PageMaxY { get; }
```

```
00543
00544 /// <summary>
00545 /// The maximum x coordinate of the content in the line.
00546 /// </summary>
00547         public double ContentMaxX { get; }
00548
00549 /// <summary>
00550 /// The minimum y coordinate of the content in the line.
00551 /// </summary>
00552         public double ContentMinY { get; }
00553
00554 /// <summary>
00555 /// The maximum y coordinate of the content in the line.
00556 /// </summary>
00557         public double ContentMaxY { get; }
00558
00559 /// <summary>
00560 /// The vertical coordinate of the baseline.
00561 /// </summary>
00562         public double Baseline { get; }
00563
00564 /// <summary>
00565 /// Create a new <see cref="LineEventArgs"/>
00566 /// </summary>
00567 /// <param name="markdownContext">The current <see
      cref="VectSharp.Markdown.MarkdownContext"/>.</param>
00568 /// <param name="graphics">The current <see cref="VectSharp.Graphics"/> surface.</param>
00569 /// <param name="line">The line that will be rendered.</param>
00570 /// <param name="pageMaxX">The maximum space available for the line on the x axis.</param>
00571 /// <param name="pageMaxY">The maximum space available for the line on the y axis.</param>
00572 /// <param name="contentMaxX">The maximum x coordinate of the content in the line.</param>
00573 /// <param name="contentMinY">The minimum y coordinate of the content in the line.</param>
00574 /// <param name="contentMaxY">The maximum y coordinate of the content in the line.</param>
00575 /// <param name="baseline">The vertical coordinate of the baseline.</param>
00576         public LineEventArgs(MarkdownContext markdownContext, Graphics graphics, Line line, double
      pageMaxX, double pageMaxY, double contentMaxX, double contentMinY, double contentMaxY, double
      baseline)
00577         {
00578             MarkdownContext = markdownContext;
00579             Graphics = graphics;
00580             Line = line;
00581             PageMaxX = pageMaxX;
00582             PageMaxY = pageMaxY;
00583             ContentMaxX = contentMaxX;
00584             ContentMinY = contentMinY;
00585             ContentMaxY = contentMaxY;
00586             Baseline = baseline;
00587         }
00588     }
00589 }
```

8.14 MarkdownRenderer.Inlines.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2024 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using Markdig.Extensions.Emoji;
00019 using Markdig.Renderers.Html;
00020 using Markdig.Syntax.Inlines;
00021 using Markdig.Syntax;
00022 using System.Collections.Generic;
00023 using System.IO;
00024 using System.Linq;
00025 using System.Text;
00026 using System;
00027 using VectSharp.Markdown.CSharpMath.VectSharp;
00028
00029 namespace VectSharp.Markdown
00030 {
```

```
00031     public partial class MarkdownRenderer
00032     {
00033         private string RenderInline(Inline inline, ref MarkdownContext context, ref Graphics graphics,
      NewPageAction newPageAction)
00034         {
00035             this.OnInlineRendering(ref context, ref graphics, ref inline);
00036 
00037             if (inline != null)
00038             {
00039                 string returnValue;
00040                 HtmlAttributes attributes = inline.TryGetAttributes();
00041 
00042                 if (attributes != null && !string.IsNullOrEmpty(attributes.Id))
00043                 {
00044                     Point cursor = context.Cursor;
00045                     RenderHTMLBlock("<a name=\"" + attributes.Id + "\"></a>", true, ref context, ref
      graphics, newPageAction, false, false);
00046                     context.Cursor = cursor;
00047                 }
00048 
00049                 if (inline is EmojiInline emoji)
00050                 {
00051                     int headingLevel = 0;
00052                     Block currBlock = emoji.Parent?.ParentBlock;
00053                     while (currBlock != null)
00054                     {
00055                         if (currBlock is HeadingBlock heading)
00056                         {
00057                             headingLevel = heading.Level;
00058                             break;
00059                         }
00060                         currBlock = currBlock.Parent;
00061                     }
00062 
00063                     inline = new LinkInline(emoji.Content.ToString() + "_heading:" +
      headingLevel.ToString(), "") { IsImage = true };
00064                 }
00065 
00066                 if (inline is LeafInline)
00067                 {
00068                     if (inline is AutolinkInline autoLink)
00069                     {
00070                         LinkInline link = new LinkInline((autoLink.IsEmail ?  "mailto:" :  "") +
      autoLink.Url, "");
00071                         link.AppendChild(new LiteralInline(autoLink.Url));
00072 
00073                         returnValue = RenderLinkInline(link, ref context, ref graphics,
      newPageAction);
00074                     }
00075                     else if (inline is CodeInline code)
00076                     {
00077                         returnValue = RenderCodeInline(code, ref context, ref graphics,
      newPageAction);
00078                     }
00079                     else if (inline is HtmlEntityInline htmlEntity)
00080                     {
00081                         returnValue = RenderLiteralInline(new LiteralInline(htmlEntity.Transcoded),
      ref context, ref graphics, newPageAction);
00082                     }
00083                     else if (inline is HtmlInline html)
00084                     {
00085                         RenderHTMLBlock(html.Tag, true, ref context, ref graphics, newPageAction,
      true, true);
00086                         returnValue = "";
00087                     }
00088                     else if (inline is LineBreakInline lineBreak)
00089                     {
00090                         RenderLineBreakInline(lineBreak.IsHard, false, ref context, ref graphics,
      newPageAction);
00091                         returnValue = "\n";
00092                     }
00093                     else if (inline is LiteralInline literal)
00094                     {
00095                         returnValue = RenderLiteralInline(literal, ref context, ref graphics,
      newPageAction);
00096                     }
00097                     else if (inline is Markdig.Extensions.Mathematics.MathInline math)
00098                     {
00099                         byte[] svgData;
00100 
00101                         using (MemoryStream ms = new MemoryStream())
00102                         {
00103                             if (math.DelimiterCount == 1)
00104                             {
00105                                 MathPainter.LineStyle = global::CSharpMath.Atom.LineStyle.Text;
00106                             }
00107                             else
```

```
00108                           {
00109                               MathPainter.LineStyle = global::CSharpMath.Atom.LineStyle.Display;
00110                           }
00111
00112                           MathPainter.FontSize = (float)(context.Font.FontSize *
      MathFontScalingFactor / ImageMultiplier);
00113
00114                           MathPainter.LaTeX = math.Content.ToString();
00115                           MathPainter.TextColor = context.Colour;
00116                           Page pag = MathPainter.DrawToPage();
00117                           VectSharp.SVG.SVGContextInterpreter.SaveAsSVG(pag, ms);
00118                           svgData = ms.ToArray();
00119                       }
00120
00121                       string imageUri = "<img src=\"" +
      Convert.ToBase64String(svgData) + "\">";
00122                       RenderHTMLBlock(imageUri, true, ref context, ref graphics, newPageAction,
      true, true);
00123                       returnValue = math.Content.ToString();
00124                   }
00125                   else if (inline is Markdig.Extensions.SmartyPants.SmartyPant smartyPant)
00126                   {
00127                       returnValue = smartyPant.Type switch
00128                       {
00129                           Markdig.Extensions.SmartyPants.SmartyPantType.LeftDoubleQuote =>
      RenderLiteralInline(new LiteralInline("“"), ref context, ref graphics, newPageAction),
00130                           Markdig.Extensions.SmartyPants.SmartyPantType.RightDoubleQuote =>
      RenderLiteralInline(new LiteralInline("”"), ref context, ref graphics, newPageAction),
00131                           Markdig.Extensions.SmartyPants.SmartyPantType.LeftQuote =>
      RenderLiteralInline(new LiteralInline("‘"), ref context, ref graphics, newPageAction),
00132                           Markdig.Extensions.SmartyPants.SmartyPantType.RightQuote =>
      RenderLiteralInline(new LiteralInline("’"), ref context, ref graphics, newPageAction),
00133                           Markdig.Extensions.SmartyPants.SmartyPantType.Dash2 =>
      RenderLiteralInline(new LiteralInline("–"), ref context, ref graphics, newPageAction),
00134                           Markdig.Extensions.SmartyPants.SmartyPantType.Dash3 =>
      RenderLiteralInline(new LiteralInline("—"), ref context, ref graphics, newPageAction),
00135                           Markdig.Extensions.SmartyPants.SmartyPantType.DoubleQuote =>
      RenderLiteralInline(new LiteralInline("\""), ref context, ref graphics, newPageAction),
00136                           Markdig.Extensions.SmartyPants.SmartyPantType.Ellipsis =>
      RenderLiteralInline(new LiteralInline("..."), ref context, ref graphics, newPageAction),
00137                           Markdig.Extensions.SmartyPants.SmartyPantType.LeftAngleQuote =>
      RenderLiteralInline(new LiteralInline("«"), ref context, ref graphics, newPageAction),
00138                           Markdig.Extensions.SmartyPants.SmartyPantType.Quote =>
      RenderLiteralInline(new LiteralInline("'"), ref context, ref graphics, newPageAction),
00139                           Markdig.Extensions.SmartyPants.SmartyPantType.RightAngleQuote =>
      RenderLiteralInline(new LiteralInline("»"), ref context, ref graphics, newPageAction),
00140                           _ => "",
00141                       };
00142                   }
00143                   else if (inline is Markdig.Extensions.TaskLists.TaskList)
00144                   {
00145                       // Nothing to render here (the checkbox has already been rendered)
00146                       returnValue = "";
00147                   }
00148                   else
00149                   {
00150                       returnValue = "";
00151                   }
00152               }
00153               else if (inline is ContainerInline)
00154               {
00155                   if (inline is DelimiterInline)
00156                   {
00157                       // Nothing to render here
00158                       returnValue = "";
00159                   }
00160                   else if (inline is EmphasisInline emphasis)
00161                   {
00162                       returnValue = RenderEmphasisInline(emphasis, ref context, ref graphics,
      newPageAction);
00163                   }
00164                   else if (inline is LinkInline link)
00165                   {
00166                       returnValue = RenderLinkInline(link, ref context, ref graphics,
      newPageAction);
00167                   }
00168                   else
00169                   {
00170                       returnValue = "";
00171                   }
00172               }
00173               else
00174               {
00175                   returnValue = "";
00176               }
00177
00178               this.OnInlineRendered(ref context, ref graphics, inline);
```

```
00179                 return returnValue;
00180             }
00181             else
00182             {
00183                 return "";
00184             }
00185         }
00186 
00187         private void RenderLineBreakInline(bool isHard, bool isPageBreak, ref MarkdownContext context,
      ref Graphics graphics, NewPageAction newPageAction)
00188         {
00189             if (isHard)
00190             {
00191                 context.CurrentLine.Render(ref graphics, ref context, newPageAction,
      this.PageSize.Height - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y,
      context.GetMaxX(context.Cursor.Y - context.Font.Ascent, context.Cursor.Y - context.Font.Descent,
      this.PageSize.Width - this.Margins.Right - context.Translation.X - context.MarginBottomRight.X),
      this);
00192                 context.CurrentLine = new Line(context.Font.Ascent);
00193                 context.Cursor = new Point(0, context.Cursor.Y - context.Font.Descent + SpaceAfterLine
      * context.Font.FontSize + context.Font.Ascent);
00194 
00195                 double minX = context.GetMinX(context.Cursor.Y - context.Font.Ascent, context.Cursor.Y
      - context.Font.Descent);
00196 
00197                 context.Cursor = new Point(minX, context.Cursor.Y);
00198 
00199                 if (isPageBreak)
00200                 {
00201                     newPageAction(ref context, ref graphics);
00202                 }
00203             }
00204             else
00205             {
00206                 double spaceWidth = context.Font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(' ') /
      1000.0 * context.Font.FontSize;
00207                 context.Cursor = new Point(context.Cursor.X + spaceWidth, context.Cursor.Y);
00208             }
00209         }
00210 
00211         private string RenderLiteralInline(LiteralInline literal, ref MarkdownContext context, ref
      Graphics graphics, NewPageAction newPageAction)
00212         {
00213             string text = literal.Content.ToString();
00214 
00215             double spaceWidth = context.Font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(' ') / 1000.0
      * context.Font.FontSize;
00216 
00217             List<Word> words = Word.GetWords(text, context.Font, this.PageSize.Width -
      this.Margins.Right - context.Translation.X - context.MarginBottomRight.X).ToList();
00218 
00219             double underlineStart = context.Cursor.X;
00220             double underlineEnd = context.Cursor.X;
00221 
00222             double currLineMaxX = context.GetMaxX(context.Cursor.Y - context.Font.Ascent,
      context.Cursor.Y - context.Font.Descent, this.PageSize.Width - this.Margins.Right -
      context.Translation.X - context.MarginBottomRight.X);
00223 
00224             bool ignoreNextWhitespace = false;
00225 
00226             bool broken = false;
00227 
00228             for (int i = 0; i < words.Count; i++)
00229             {
00230                 Word w = words[i];
00231 
00232                 if (!string.IsNullOrEmpty(w.Text))
00233                 {
00234                     if (!ignoreNextWhitespace)
00235                     {
00236                         context.Cursor = new Point(context.Cursor.X + spaceWidth * w.WhitespaceCount *
      (w.PrecedingWhitespace == '\t' ?  4 :  1), context.Cursor.Y);
00237                     }
00238                     else
00239                     {
00240                         ignoreNextWhitespace = false;
00241                     }
00242 
00243                     Font.DetailedFontMetrics wordMetrics = w.Metrics;
00244 
00245                     double finalX = context.Cursor.X + wordMetrics.Width +
      wordMetrics.RightSideBearing + wordMetrics.LeftSideBearing;
00246 
00247                     if (finalX <= currLineMaxX || broken)
00248                     {
00249                         context.CurrentLine.Fragments.Add(new TextFragment(new Point(context.Cursor.X
      + wordMetrics.LeftSideBearing, context.Cursor.Y), w.Text, context.Font, context.Colour, context.Tag));
```

```
00250                         context.Cursor = new Point(context.Cursor.X + wordMetrics.Width +
      wordMetrics.RightSideBearing + wordMetrics.LeftSideBearing, context.Cursor.Y);
00251
00252                         broken = false;
00253
00254                         if (context.Underline || context.StrikeThrough)
00255                         {
00256                             underlineEnd = context.Cursor.X;
00257                         }
00258                     }
00259                     else
00260                     {
00261                         if (context.Underline && underlineStart != underlineEnd)
00262                         {
00263                             context.CurrentLine.Fragments.Add(new UnderlineFragment(new
      Point(underlineStart, context.Cursor.Y + context.Font.FontSize * 0.2), new Point(underlineEnd,
      context.Cursor.Y + context.Font.FontSize * 0.2), context.Colour, context.Font.FontSize *
      (context.Font.FontFamily.IsBold ?  this.BoldUnderlineThickness :  this.UnderlineThickness),
      context.Tag));
00264                         }
00265                         else if (context.StrikeThrough && underlineStart != underlineEnd)
00266                         {
00267                             context.CurrentLine.Fragments.Add(new UnderlineFragment(new
      Point(underlineStart, context.Cursor.Y - context.Font.Ascent * 0.5 - context.Font.Descent * 0.5), new
      Point(underlineEnd, context.Cursor.Y - context.Font.Ascent * 0.5 - context.Font.Descent * 0.5),
      context.Colour, context.Font.FontSize * (context.Font.FontFamily.IsBold ?  this.BoldUnderlineThickness
      :  this.UnderlineThickness), context.Tag));
00268                         }
00269
00270                         context.CurrentLine.Render(ref graphics, ref context, newPageAction,
      this.PageSize.Height - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y,
      context.GetMaxX(context.Cursor.Y - context.Font.Ascent, context.Cursor.Y - context.Font.Descent,
      this.PageSize.Width - this.Margins.Right - context.Translation.X - context.MarginBottomRight.X),
      this);
00271
00272
00273
00274                         context.CurrentLine = new Line(context.Font.Ascent);
00275                         context.Cursor = new Point(0, context.Cursor.Y - context.Font.Descent +
      SpaceAfterLine * context.Font.FontSize + context.Font.Ascent);
00276                         currLineMaxX = context.GetMaxX(context.Cursor.Y - context.Font.Ascent,
      context.Cursor.Y - context.Font.Descent, this.PageSize.Width - this.Margins.Right -
      context.Translation.X - context.MarginBottomRight.X);
00277
00278                         double minX = context.GetMinX(context.Cursor.Y - context.Font.Ascent,
      context.Cursor.Y - context.Font.Descent);
00279
00280                         context.Cursor = new Point(minX, context.Cursor.Y);
00281
00282                         underlineStart = minX;
00283                         underlineEnd = minX;
00284
00285                         i--;
00286                         ignoreNextWhitespace = true;
00287                         broken = true;
00288                     }
00289                 }
00290                 else
00291                 {
00292                     context.Cursor = new Point(context.Cursor.X + spaceWidth * w.WhitespaceCount *
      (w.PrecedingWhitespace == '\t' ?  4 :  1), context.Cursor.Y);
00293                 }
00294             }
00295
00296             if (context.Underline && underlineStart != underlineEnd)
00297             {
00298                 context.CurrentLine.Fragments.Add(new UnderlineFragment(new Point(underlineStart,
      context.Cursor.Y + context.Font.FontSize * 0.2), new Point(underlineEnd, context.Cursor.Y +
      context.Font.FontSize * 0.2), context.Colour, context.Font.FontSize * (context.Font.FontFamily.IsBold
      ?  this.BoldUnderlineThickness :  this.UnderlineThickness), context.Tag));
00299             }
00300             else if (context.StrikeThrough && underlineStart != underlineEnd)
00301             {
00302                 context.CurrentLine.Fragments.Add(new UnderlineFragment(new Point(underlineStart,
      context.Cursor.Y - context.Font.Ascent * 0.5 - context.Font.Descent * 0.5), new Point(underlineEnd,
      context.Cursor.Y - context.Font.Ascent * 0.5 - context.Font.Descent * 0.5), context.Colour,
      context.Font.FontSize * (context.Font.FontFamily.IsBold ?  this.BoldUnderlineThickness :
      this.UnderlineThickness), context.Tag));
00303             }
00304
00305             return text;
00306         }
00307
00308         private string RenderCodeInline(CodeInline code, ref MarkdownContext context, ref Graphics
      graphics, NewPageAction newPageAction)
00309         {
00310             MarkdownContext prevContext = context.Clone();
```

```
00311
00312             context.Font = new Font(this.CodeFont, context.Font.FontSize);
00313
00314             double spaceWidth = context.Font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(' ') / 1000.0
      * context.Font.FontSize;
00315
00316             string text = code.Content.ToString();
00317             List<Word> words = Word.GetWords(text, context.Font, this.PageSize.Width -
      this.Margins.Right - context.Translation.X - context.MarginBottomRight.X).ToList();
00318
00319             double currLineMaxX = context.GetMaxX(context.Cursor.Y - context.Font.Ascent,
      context.Cursor.Y - context.Font.Descent, this.PageSize.Width - this.Margins.Right -
      context.Translation.X - context.MarginBottomRight.X);
00320
00321             context.Cursor = new Point(context.Cursor.X + this.CodeInlineMargin *
      context.Font.FontSize, context.Cursor.Y);
00322
00323
00324             double underlineStart = context.Cursor.X;
00325             double underlineEnd = context.Cursor.X;
00326
00327             bool broken = false;
00328
00329             for (int i = 0; i < words.Count; i++)
00330             {
00331                 Word w = words[i];
00332
00333                 if (!string.IsNullOrEmpty(w.Text))
00334                 {
00335                     context.Cursor = new Point(context.Cursor.X + spaceWidth * w.WhitespaceCount *
      (w.PrecedingWhitespace == '\t' ?  4 :  1), context.Cursor.Y);
00336
00337                     Font.DetailedFontMetrics wordMetrics = w.Metrics;
00338
00339                     double finalX = context.Cursor.X + wordMetrics.Width +
      wordMetrics.RightSideBearing + wordMetrics.LeftSideBearing;
00340
00341                     if (finalX <= currLineMaxX || broken)
00342                     {
00343                         if (i == 0)
00344                         {
00345                             context.CurrentLine.Fragments.Add(new RectangleFragment(new
      Point(context.Cursor.X - this.CodeInlineMargin * context.Font.FontSize, context.Cursor.Y -
      context.Font.YMax), new Size(this.CodeInlineMargin * context.Font.FontSize, context.Font.YMax -
      context.Font.YMin), CodeInlineBackgroundColour, context.Tag));
00346                         }
00347
00348                         context.CurrentLine.Fragments.Add(new RectangleFragment(new
      Point(context.Cursor.X - spaceWidth * w.WhitespaceCount * (w.PrecedingWhitespace == '\t' ?  4 :  1),
      context.Cursor.Y - context.Font.YMax), new Size(wordMetrics.Width + wordMetrics.LeftSideBearing * 2 +
      wordMetrics.RightSideBearing + spaceWidth * w.WhitespaceCount * (w.PrecedingWhitespace == '\t' ?  4 :
      1), context.Font.YMax - context.Font.YMin), CodeInlineBackgroundColour, context.Tag));
00349
00350                         context.CurrentLine.Fragments.Add(new TextFragment(new Point(context.Cursor.X
      + wordMetrics.LeftSideBearing, context.Cursor.Y), w.Text, context.Font, context.Colour, context.Tag));
00351                         context.Cursor = new Point(context.Cursor.X + wordMetrics.Width +
      wordMetrics.RightSideBearing + wordMetrics.LeftSideBearing, context.Cursor.Y);
00352
00353                         broken = false;
00354
00355                         if (context.Underline || context.StrikeThrough)
00356                         {
00357                             underlineEnd = context.Cursor.X;
00358                         }
00359                     }
00360                     else
00361                     {
00362                         if (context.Underline && underlineStart != underlineEnd)
00363                         {
00364                             context.CurrentLine.Fragments.Add(new UnderlineFragment(new
      Point(underlineStart, context.Cursor.Y + context.Font.FontSize * 0.2), new Point(underlineEnd,
      context.Cursor.Y + context.Font.FontSize * 0.2), context.Colour, context.Font.FontSize *
      (context.Font.FontFamily.IsBold ?  this.BoldUnderlineThickness :  this.UnderlineThickness),
      context.Tag));
00365                         }
00366                         else if (context.StrikeThrough && underlineStart != underlineEnd)
00367                         {
00368                             context.CurrentLine.Fragments.Add(new UnderlineFragment(new
      Point(underlineStart, context.Cursor.Y - context.Font.Ascent * 0.5 - context.Font.Descent * 0.5), new
      Point(underlineEnd, context.Cursor.Y - context.Font.Ascent * 0.5 - context.Font.Descent * 0.5),
      context.Colour, context.Font.FontSize * (context.Font.FontFamily.IsBold ?  this.BoldUnderlineThickness
      :  this.UnderlineThickness), context.Tag));
00369                         }
00370
00371                         context.CurrentLine.Render(ref graphics, ref context, newPageAction,
      this.PageSize.Height - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y,
      context.GetMaxX(context.Cursor.Y - context.Font.Ascent, context.Cursor.Y - context.Font.Descent,
```

```
    this.PageSize.Width - this.Margins.Right - context.Translation.X - context.MarginBottomRight.X),
    this);
00372
00373                       context.CurrentLine = new Line(prevContext.Font.Ascent);
00374
00375                       context.Cursor = new Point(0, context.Cursor.Y - prevContext.Font.Descent +
    SpaceAfterLine * prevContext.Font.FontSize + prevContext.Font.Ascent);
00376
00377
00378                       double minX = context.GetMinX(context.Cursor.Y - context.Font.Ascent,
    context.Cursor.Y - context.Font.Descent);
00379
00380                       context.Cursor = new Point(minX, context.Cursor.Y);
00381
00382                       if (i == 0)
00383                       {
00384                           context.Cursor = new Point(context.Cursor.X + this.CodeInlineMargin *
    context.Font.FontSize, context.Cursor.Y);
00385                       }
00386
00387                       underlineStart = minX;
00388                       underlineEnd = minX;
00389
00390                       i--;
00391                       broken = true;
00392                   }
00393               }
00394           }
00395
00396
00397           if (context.Underline && underlineStart != underlineEnd)
00398           {
00399               context.CurrentLine.Fragments.Add(new UnderlineFragment(new Point(underlineStart,
    context.Cursor.Y + context.Font.FontSize * 0.2), new Point(underlineEnd, context.Cursor.Y +
    context.Font.FontSize * 0.2), context.Colour, context.Font.FontSize * (context.Font.FontFamily.IsBold
    ?  this.BoldUnderlineThickness :  this.UnderlineThickness), context.Tag));
00400           }
00401           else if (context.StrikeThrough && underlineStart != underlineEnd)
00402           {
00403               context.CurrentLine.Fragments.Add(new UnderlineFragment(new Point(underlineStart,
    context.Cursor.Y - context.Font.Ascent * 0.5 - context.Font.Descent * 0.5), new Point(underlineEnd,
    context.Cursor.Y - context.Font.Ascent * 0.5 - context.Font.Descent * 0.5), context.Colour,
    context.Font.FontSize * (context.Font.FontFamily.IsBold ?  this.BoldUnderlineThickness :
    this.UnderlineThickness), context.Tag));
00404           }
00405
00406           context.CurrentLine.Fragments.Add(new RectangleFragment(new Point(context.Cursor.X,
    context.Cursor.Y - context.Font.YMax), new Size(this.CodeInlineMargin * context.Font.FontSize,
    context.Font.YMax - context.Font.YMin), CodeInlineBackgroundColour, context.Tag));
00407
00408           context.Cursor = new Point(context.Cursor.X + this.CodeInlineMargin *
    context.Font.FontSize, context.Cursor.Y);
00409
00410           prevContext.Cursor = context.Cursor;
00411           prevContext.BottomRight = context.BottomRight;
00412           prevContext.CurrentPage = context.CurrentPage;
00413           prevContext.CurrentLine = context.CurrentLine;
00414
00415           context = prevContext;
00416           return text;
00417       }
00418
00419       private void RenderCodeBlockLine(string text, ref MarkdownContext context, ref Graphics
    graphics, NewPageAction newPageAction)
00420       {
00421           double spaceWidth = context.Font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(' ') / 1000.0
    * context.Font.FontSize;
00422
00423           List<Word> words = Word.GetWords(text, context.Font, this.PageSize.Width -
    this.Margins.Right - context.Translation.X - context.Font.FontSize * 2 -
    context.MarginBottomRight.X).ToList();
00424
00425           double underlineStart = context.Cursor.X;
00426           double underlineEnd = context.Cursor.X;
00427
00428           double minX = context.GetMinX(context.Cursor.Y - context.Font.Ascent, context.Cursor.Y -
    context.Font.Descent);
00429
00430           double currLineMaxX = context.GetMaxX(context.Cursor.Y - context.Font.Ascent,
    context.Cursor.Y - context.Font.Descent, this.PageSize.Width - this.Margins.Right -
    context.Translation.X - context.MarginBottomRight.X) - context.Font.FontSize;
00431
00432           bool broken = false;
00433
00434           for (int i = 0; i < words.Count; i++)
00435           {
00436               Word w = words[i];
```

```
00437                 if (!string.IsNullOrEmpty(w.Text))
00438                 {
00439                     context.Cursor = new Point(context.Cursor.X + spaceWidth * w.WhitespaceCount *
      (w.PrecedingWhitespace == '\t' ?  4 :  1), context.Cursor.Y);
00440                     Font.DetailedFontMetrics wordMetrics = w.Metrics;
00441 
00442                     double finalX = context.Cursor.X + wordMetrics.Width +
      wordMetrics.RightSideBearing + wordMetrics.LeftSideBearing;
00443 
00444                     double effW = wordMetrics.Width + wordMetrics.RightSideBearing +
      wordMetrics.LeftSideBearing;
00445 
00446                     double maxW = this.PageSize.Width - this.Margins.Right - context.Translation.X -
      context.Font.FontSize * 2 - spaceWidth * w.WhitespaceCount * (w.PrecedingWhitespace == '\t' ?  4 :  1)
      - context.MarginBottomRight.X;
00447 
00448                     if (finalX <= currLineMaxX || broken)
00449                     {
00450                         broken = false;
00451                         context.CurrentLine.Fragments.Add(new TextFragment(new Point(context.Cursor.X
      + wordMetrics.LeftSideBearing, context.Cursor.Y), w.Text, context.Font, context.Colour, context.Tag));
00452                         context.Cursor = new Point(context.Cursor.X + wordMetrics.Width +
      wordMetrics.RightSideBearing + wordMetrics.LeftSideBearing, context.Cursor.Y);
00453 
00454                         if (context.Underline || context.StrikeThrough)
00455                         {
00456                             underlineEnd = context.Cursor.X;
00457                         }
00458                     }
00459                     else
00460                     {
00461                         context.CurrentLine.Fragments.Insert(0, new RectangleFragment(new Point(minX,
      context.Cursor.Y - context.Font.YMax), new Size(currLineMaxX + context.Font.FontSize - minX,
      context.Font.YMax - context.Font.YMin + this.SpaceAfterLine * context.Font.FontSize),
      CodeBlockBackgroundColour, context.Tag));
00462 
00463                         if (context.Underline && underlineStart != underlineEnd)
00464                         {
00465                             context.CurrentLine.Fragments.Add(new UnderlineFragment(new
      Point(underlineStart, context.Cursor.Y + context.Font.FontSize * 0.2), new Point(underlineEnd,
      context.Cursor.Y + context.Font.FontSize * 0.2), context.Colour, context.Font.FontSize *
      (context.Font.FontFamily.IsBold ?  this.BoldUnderlineThickness :  this.UnderlineThickness),
      context.Tag));
00466                         }
00467                         else if (context.StrikeThrough && underlineStart != underlineEnd)
00468                         {
00469                             context.CurrentLine.Fragments.Add(new UnderlineFragment(new
      Point(underlineStart, context.Cursor.Y - context.Font.Ascent * 0.5 - context.Font.Descent * 0.5), new
      Point(underlineEnd, context.Cursor.Y - context.Font.Ascent * 0.5 - context.Font.Descent * 0.5),
      context.Colour, context.Font.FontSize * (context.Font.FontFamily.IsBold ?  this.BoldUnderlineThickness
      :  this.UnderlineThickness), context.Tag));
00470                         }
00471 
00472                         context.CurrentLine.Render(ref graphics, ref context, newPageAction,
      this.PageSize.Height - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y,
      context.GetMaxX(context.Cursor.Y - context.Font.Ascent, context.Cursor.Y - context.Font.Descent,
      this.PageSize.Width - this.Margins.Right - context.Translation.X - context.MarginBottomRight.X),
      this);
00473 
00474                         underlineStart = 0;
00475                         underlineEnd = 0;
00476 
00477                         context.CurrentLine = new Line(context.Font.Ascent);
00478 
00479                         context.Cursor = new Point(0, context.Cursor.Y - context.Font.Descent +
      SpaceAfterLine * context.Font.FontSize + context.Font.Ascent);
00480 
00481                         currLineMaxX = context.GetMaxX(context.Cursor.Y - context.Font.Ascent,
      context.Cursor.Y - context.Font.Descent, this.PageSize.Width - this.Margins.Right -
      context.Translation.X - context.MarginBottomRight.X) - context.Font.FontSize;
00482 
00483                         minX = context.GetMinX(context.Cursor.Y - context.Font.Ascent,
      context.Cursor.Y - context.Font.Descent);
00484 
00485                         context.Cursor = new Point(minX + context.Font.FontSize, context.Cursor.Y);
00486 
00487                         i--;
00488                         broken = true;
00489                     }
00490                 }
00491                 else
00492                 {
00493                     context.Cursor = new Point(context.Cursor.X + spaceWidth * w.WhitespaceCount *
      (w.PrecedingWhitespace == '\t' ?  4 :  1), context.Cursor.Y);
00494                 }
00495             }
00496 
```

```
00497             if (context.Underline && underlineStart != underlineEnd)
00498             {
00499                 context.CurrentLine.Fragments.Add(new UnderlineFragment(new Point(underlineStart,
      context.Cursor.Y + context.Font.FontSize * 0.2), new Point(underlineEnd, context.Cursor.Y +
      context.Font.FontSize * 0.2), context.Colour, context.Font.FontSize * (context.Font.FontFamily.IsBold
      ?  this.BoldUnderlineThickness :  this.UnderlineThickness), context.Tag));
00500             }
00501             else if (context.StrikeThrough && underlineStart != underlineEnd)
00502             {
00503                 context.CurrentLine.Fragments.Add(new UnderlineFragment(new Point(underlineStart,
      context.Cursor.Y - context.Font.Ascent * 0.5 - context.Font.Descent * 0.5), new Point(underlineEnd,
      context.Cursor.Y - context.Font.Ascent * 0.5 - context.Font.Descent * 0.5), context.Colour,
      context.Font.FontSize * (context.Font.FontFamily.IsBold ?  this.BoldUnderlineThickness :
      this.UnderlineThickness), context.Tag));
00504             }
00505
00506             context.CurrentLine.Fragments.Insert(0, new RectangleFragment(new Point(minX,
      context.Cursor.Y - context.Font.YMax), new Size(currLineMaxX + context.Font.FontSize - minX,
      context.Font.YMax - context.Font.YMin + this.SpaceAfterLine * context.Font.FontSize),
      CodeBlockBackgroundColour, context.Tag));
00507         }
00508
00509         private string RenderEmphasisInline(EmphasisInline emphasis, ref MarkdownContext context, ref
      Graphics graphics, NewPageAction newPageAction)
00510         {
00511             MarkdownContext prevContext = context.Clone();
00512
00513             Point translationToUndo = new Point(0, 0);
00514
00515             switch (emphasis.DelimiterChar)
00516             {
00517                 case '*':
00518                 case '_':
00519                     if (emphasis.DelimiterCount == 2)
00520                     {
00521                         if (context.Font.FontFamily == this.ItalicFontFamily)
00522                         {
00523                             context.Font = new Font(this.BoldItalicFontFamily, context.Font.FontSize);
00524                         }
00525                         else if (context.Font.FontFamily == this.BoldFontFamily)
00526                         {
00527                             context.Font = new Font(this.RegularFontFamily, context.Font.FontSize);
00528                         }
00529                         else if (context.Font.FontFamily == this.BoldItalicFontFamily)
00530                         {
00531                             context.Font = new Font(this.ItalicFontFamily, context.Font.FontSize);
00532                         }
00533                         else
00534                         {
00535                             context.Font = new Font(this.BoldFontFamily, context.Font.FontSize);
00536                         }
00537                     }
00538                     else if (emphasis.DelimiterCount == 3)
00539                     {
00540                         if (context.Font.FontFamily == this.ItalicFontFamily)
00541                         {
00542                             context.Font = new Font(this.BoldFontFamily, context.Font.FontSize);
00543                         }
00544                         else if (context.Font.FontFamily == this.BoldFontFamily)
00545                         {
00546                             context.Font = new Font(this.ItalicFontFamily, context.Font.FontSize);
00547                         }
00548                         else if (context.Font.FontFamily == this.BoldItalicFontFamily)
00549                         {
00550                             context.Font = new Font(this.RegularFontFamily, context.Font.FontSize);
00551                         }
00552                         else
00553                         {
00554                             context.Font = new Font(this.BoldItalicFontFamily, context.Font.FontSize);
00555                         }
00556                     }
00557                     else
00558                     {
00559                         if (context.Font.FontFamily == this.ItalicFontFamily)
00560                         {
00561                             context.Font = new Font(this.RegularFontFamily, context.Font.FontSize);
00562                         }
00563                         else if (context.Font.FontFamily == this.BoldFontFamily)
00564                         {
00565                             context.Font = new Font(this.BoldItalicFontFamily, context.Font.FontSize);
00566                         }
00567                         else if (context.Font.FontFamily == this.BoldItalicFontFamily)
00568                         {
00569                             context.Font = new Font(this.BoldFontFamily, context.Font.FontSize);
00570                         }
00571                         else
00572                         {
```

```
00573                               context.Font = new Font(this.ItalicFontFamily, context.Font.FontSize);
00574                           }
00575                       }
00576                       break;
00577                   case '"':
00578                       if (emphasis.DelimiterCount == 2)
00579                       {
00580                           if (context.Font.FontFamily == this.ItalicFontFamily)
00581                           {
00582                               context.Font = new Font(this.RegularFontFamily, context.Font.FontSize);
00583                           }
00584                           else if (context.Font.FontFamily == this.BoldFontFamily)
00585                           {
00586                               context.Font = new Font(this.BoldItalicFontFamily, context.Font.FontSize);
00587                           }
00588                           else if (context.Font.FontFamily == this.BoldItalicFontFamily)
00589                           {
00590                               context.Font = new Font(this.BoldFontFamily, context.Font.FontSize);
00591                           }
00592                           else
00593                           {
00594                               context.Font = new Font(this.ItalicFontFamily, context.Font.FontSize);
00595                           }
00596                       }
00597                       break;
00598                   case '~':
00599                       if (emphasis.DelimiterCount == 1)
00600                       {
00601                           //subscript;
00602                           context.Cursor = new Point(context.Cursor.X, context.Cursor.Y +
      context.Font.FontSize * this.SubscriptShift);
00603                           translationToUndo = new Point(translationToUndo.X, translationToUndo.Y +
      context.Font.FontSize * this.SubscriptShift);
00604                           context.Font = new Font(context.Font.FontFamily, context.Font.FontSize *
      this.SubSuperscriptFontSize);
00605                       }
00606                       else
00607                       {
00608                           //strikethrough
00609                           context.StrikeThrough = true;
00610                       }
00611                       break;
00612                   case '^':
00613                       if (emphasis.DelimiterCount == 1)
00614                       {
00615                           //superscript
00616                           context.Cursor = new Point(context.Cursor.X, context.Cursor.Y -
      context.Font.FontSize * this.SuperscriptShift);
00617                           translationToUndo = new Point(translationToUndo.X, translationToUndo.Y -
      context.Font.FontSize * this.SuperscriptShift);
00618                           context.Font = new Font(context.Font.FontFamily, context.Font.FontSize *
      this.SubSuperscriptFontSize);
00619                       }
00620                       break;
00621                   case '+':
00622                       context.Colour = this.InsertedColour;
00623                       break;
00624                   case '=':
00625                       context.Colour = this.MarkedColour;
00626                       break;
00627               }
00628
00629               StringBuilder text = new StringBuilder();
00630
00631               foreach (Inline innerInline in emphasis)
00632               {
00633                   text.Append(RenderInline(innerInline, ref context, ref graphics, newPageAction));
00634               }
00635
00636               if (translationToUndo.X != 0 || translationToUndo.Y != 0)
00637               {
00638                   context.Cursor = new Point(context.Cursor.X - translationToUndo.X, context.Cursor.Y -
      translationToUndo.Y);
00639               }
00640
00641
00642               prevContext.Cursor = context.Cursor;
00643               prevContext.BottomRight = context.BottomRight;
00644               prevContext.CurrentPage = context.CurrentPage;
00645               prevContext.CurrentLine = context.CurrentLine;
00646
00647               context = prevContext;
00648
00649               return text.ToString();
00650           }
00651
00652           private string RenderLinkInline(LinkInline link, ref MarkdownContext context, ref Graphics
```

```
      graphics, NewPageAction newPageAction)
00653         {
00654             if (!link.IsImage)
00655             {
00656                 MarkdownContext prevContext = context.Clone();
00657 
00658                 context.Colour = this.LinkColour;
00659                 context.Underline = true;
00660                 string tag = Guid.NewGuid().ToString("N");
00661 
00662                 if (!link.Url.StartsWith("#"))
00663                 {
00664                     if (Uri.TryCreate(this.BaseLinkUri, link.Url, out Uri uri))
00665                     {
00666                         context.LinkDestinations[tag] = this.LinkUriResolver(uri.ToString());
00667                     }
00668                     else
00669                     {
00670                         context.LinkDestinations[tag] = this.LinkUriResolver(link.Url);
00671                     }
00672                 }
00673                 else
00674                 {
00675                     if (!context.InternalAnchors.TryGetValue(link.Url, out string anchor))
00676                     {
00677                         anchor = Guid.NewGuid().ToString("N");
00678                         context.InternalAnchors[link.Url] = anchor;
00679                     }
00680 
00681                     context.LinkDestinations[tag] = "#" + anchor;
00682                 }
00683 
00684                 context.Tag = tag;
00685 
00686                 StringBuilder text = new StringBuilder();
00687 
00688                 foreach (Inline innerInline in link)
00689                 {
00690                     text.Append(RenderInline(innerInline, ref context, ref graphics, newPageAction));
00691                 }
00692 
00693                 prevContext.Cursor = context.Cursor;
00694                 prevContext.BottomRight = context.BottomRight;
00695                 prevContext.CurrentPage = context.CurrentPage;
00696                 prevContext.CurrentLine = context.CurrentLine;
00697 
00698                 context = prevContext;
00699 
00700                 return text.ToString();
00701             }
00702             else
00703             {
00704                 HtmlTag tag = HtmlTag.Parse("<img src=\"" + link.Url.Replace("\"", "\\\"") +
      "\">").FirstOrDefault();
00705 
00706                 RenderHTMLImage(tag, true, ref context, ref graphics, newPageAction);
00707 
00708                 return "";
00709             }
00710         }
00711     }
00712 }
```

8.15 MarkdownRenderer.Properties.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2024 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using Markdig;
```

```
00019 using Markdig.Extensions.Emoji;
00020 using System;
00021 using System.Collections.Generic;
00022 using System.Linq;
00023
00024 namespace VectSharp.Markdown
00025 {
00026     public partial class MarkdownRenderer
00027     {
00028 /// <summary>
00029 /// The base font size to use when rendering the document.  This will be the size of regular elements,
      and the size of header elements will be expressed as a multiple of this.
00030 /// </summary>
00031         public double BaseFontSize { get; set; } = 9.71424;
00032
00033 /// <summary>
00034 /// Scaling factor for the font size to use when rendering math.
00035 /// </summary>
00036         public double MathFontScalingFactor { get; set; } = 0.85;
00037
00038 /// <summary>
00039 /// The font size for elements at each header level.  The values in this array will be multiplied by
      the <see cref="BaseFontSize"/>.
00040 /// </summary>
00041         public double[] HeaderFontSizeMultipliers { get; } = new double[]
00042         {
00043             28 / 12.0, 22 / 12.0, 16 / 12.0, 14 / 12.0, 13 / 12.0, 12 / 12.0
00044         };
00045
00046 /// <summary>
00047 /// The thickness of the separator line after a header of each level.  A value of 0 disables the line
      after headers of that level.
00048 /// </summary>
00049         public double[] HeaderLineThicknesses { get; } = new double[] { 1, 1, 0, 0, 0, 0 };
00050
00051 /// <summary>
00052 /// The thickness of thematic break lines.
00053 /// </summary>
00054         public double ThematicBreakThickness { get; set; } = 2;
00055
00056 /// <summary>
00057 /// The font family for regular text.
00058 /// </summary>
00059         public FontFamily RegularFontFamily { get; set; } =
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica);
00060
00061 /// <summary>
00062 /// The font family for bold text.
00063 /// </summary>
00064         public FontFamily BoldFontFamily { get; set; } =
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold);
00065
00066 /// <summary>
00067 /// The font family for italic text.
00068 /// </summary>
00069         public FontFamily ItalicFontFamily { get; set; } =
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaOblique);
00070
00071 /// <summary>
00072 /// The font family for bold italic text.
00073 /// </summary>
00074         public FontFamily BoldItalicFontFamily { get; set; } =
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBoldOblique);
00075
00076 /// <summary>
00077 /// The font family for code elements.
00078 /// </summary>
00079         public FontFamily CodeFont { get; set; } =
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Courier);
00080
00081 /// <summary>
00082 /// The font family for bold code elements.
00083 /// </summary>
00084         public FontFamily CodeFontBold { get; set; } =
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierBold);
00085
00086 /// <summary>
00087 /// The font family for italic code elements.
00088 /// </summary>
00089         public FontFamily CodeFontItalic { get; set; } =
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierOblique);
00090
00091 /// <summary>
00092 /// The font family for bold italic code elements.
00093 /// </summary>
00094         public FontFamily CodeFontBoldItalic { get; set; } =
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierBoldOblique);
```

```
00095
00096 /// <summary>
00097 /// The thickness of underlines.  This value will be multiplied by the font size of the element being
      underlined.
00098 /// </summary>
00099         public double UnderlineThickness { get; set; } = 0.075;
00100
00101 /// <summary>
00102 /// The thickness of underlines for bold text.  This value will be multiplied by the font size of the
      element being underlined.
00103 /// </summary>
00104         public double BoldUnderlineThickness { get; set; } = 0.15;
00105
00106 /// <summary>
00107 /// The margins of the page.
00108 /// </summary>
00109         public Margins Margins { get; set; } = new Margins(55, 55, 55, 55);
00110
00111 /// <summary>
00112 /// The margins for table cells.
00113 /// </summary>
00114         public Margins TableCellMargins { get; set; } = new Margins(5, 0, 5, 0);
00115
00116 /// <summary>
00117 /// Defines the options for the vertical alignment of table cells.
00118 /// </summary>
00119         public enum VerticalAlignment
00120         {
00121 /// <summary>
00122 /// Table cells will be aligned at the top of their row.
00123 /// </summary>
00124             Top,
00125
00126 /// <summary>
00127 /// Table cells will be aligned in the middle of their row.
00128 /// </summary>
00129             Middle,
00130
00131 /// <summary>
00132 /// Table cells will be aligned at the bottom of their row.
00133 /// </summary>
00134             Bottom
00135         }
00136
00137 /// <summary>
00138 /// The vertical alignment of table cells.
00139 /// </summary>
00140         public VerticalAlignment TableVAlign { get; set; } = VerticalAlignment.Middle;
00141
00142 /// <summary>
00143 /// The size of the page.
00144 /// </summary>
00145         public Size PageSize { get; set; } = new Size(595, 842);
00146
00147 /// <summary>
00148 /// The space before each text paragraph.  This value will be multiplied by the <see
      cref="BaseFontSize"/>.
00149 /// </summary>
00150         public double SpaceBeforeParagaph { get; set; } = 0;
00151
00152 /// <summary>
00153 /// The space after each text paragraph.  This value will be multiplied by the <see
      cref="BaseFontSize"/>.
00154 /// </summary>
00155         public double SpaceAfterParagraph { get; set; } = 0.75;
00156
00157 /// <summary>
00158 /// The space after each line of text.  This value will be multiplied by the <see
      cref="BaseFontSize"/>.
00159 /// </summary>
00160         public double SpaceAfterLine { get; set; } = 0.25;
00161
00162 /// <summary>
00163 /// The space before each heading.  This value will be multiplied by the font size of the heading.
00164 /// </summary>
00165         public double SpaceBeforeHeading { get; set; } = 0.25;
00166
00167 /// <summary>
00168 /// The space after each heading.  This value will be multiplied by the font size of the heading.
00169 /// </summary>
00170         public double SpaceAfterHeading { get; set; } = 0.25;
00171
00172 /// <summary>
00173 /// The margin at the left and right of code inlines.  This value will be multiplied by the current
      font size.
00174 /// </summary>
00175         public double CodeInlineMargin { get; set; } = 0.25;
```

```
00176
00177 /// <summary>
00178 /// The indentation width used for list items.
00179 /// </summary>
00180         public double IndentWidth { get; set; } = 40;
00181
00182 /// <summary>
00183 /// The indentation width used for block quotes.
00184 /// </summary>
00185         public double QuoteBlockIndentWidth { get; set; } = 30;
00186
00187 /// <summary>
00188 /// The thickness of the bar to the left of block quotes.
00189 /// </summary>
00190         public double QuoteBlockBarWidth { get; set; } = 5;
00191
00192 /// <summary>
00193 /// The font size for subscripts and superscripts.  This value will be multiplied by the current font
      size.
00194 /// </summary>
00195         public double SubSuperscriptFontSize { get; set; } = 0.7;
00196
00197 /// <summary>
00198 /// The upwards shift in the baseline for superscript elements.  This value will be multiplied by the
      current font size.
00199 /// </summary>
00200         public double SuperscriptShift { get; set; } = 0.33;
00201
00202 /// <summary>
00203 /// The downwards shift in the baseline for subscript elements.  This value will be multiplied by the
      current font size.
00204 /// </summary>
00205         public double SubscriptShift { get; set; } = 0.14;
00206
00207 /// <summary>
00208 /// The base uri for resolving relative image addresses.
00209 /// </summary>
00210         public string BaseImageUri { get; set; } = "";
00211
00212 /// <summary>
00213 /// A method used to resolve (possibly remote) image uris into local file paths.  The first argument
      of the method should be the image uri and the second argument the base uri used to resolve relative
      links.  The method should return a tuple containing the path of the local file and a boolean value
      indicating whether the file has been fetched from a remote location and should be deleted after the
      program has finished using it.
00214 /// </summary>
00215         public Func<string, string, (string, bool)> ImageUriResolver { get; set; } =
      HTTPUtils.ResolveImageURI;
00216
00217 /// <summary>
00218 /// A method used to resolve emojis.  The argument of the method should be an emoji uri of the form
      "emoji://{name}_heading:{level}", where "{name}" is the name of the emoji and "{level}" is the heading
      level, or "unicode://{unicode}_heading:{level}", where "{unicode}" is a unicode surrogate pair.  The
      method should return a tuple containing the path of a local file containing the rendered emoji and a
      boolean value indicating whether the file should be deleted after the program has finished using it.
00219 /// </summary>
00220         public Func<string, (string, bool)> EmojiUriResolver { get; set; }
00221
00222 /// <summary>
00223 /// The base uri for resolving links.
00224 /// </summary>
00225         public Uri BaseLinkUri { get; set; } = new Uri("about:blank");
00226
00227 /// <summary>
00228 /// A method used to resolve link addresses.  The argument of the method should be the absolute link,
      and the method should return the resolved address.  This can be used to "redirect" links to a
      different target.
00229 /// </summary>
00230         public Func<string, string> LinkUriResolver { get; set; } = a => a;
00231
00232 /// <summary>
00233 /// A method used to a load raster image from a local file.  The argument of the method should be the
      path of a local image file, and the method should return a RasterImage representing that file.  For
      example, this can be achieved using the <c>RasterImageFile</c> class from the
      <c>VectSharp.MuPDFUtils</c> package.  If this is <see langword="null" />, only SVG images will be
      included in the document.
00234 /// </summary>
00235         public Func<string, RasterImage> RasterImageLoader { get; set; } = null;
00236
00237 /// <summary>
00238 /// The size of images (as defined in the image's width and height attributes) will be multiplied by
      this value to determine the actual size of the image on the page.  This has no effect on images
      without a width or height attribute.
00239 /// </summary>
00240         public double ImageUnitMultiplier { get; set; } = 0.60714;
00241
00242 /// <summary>
```

```
00243 /// The size of images will be multiplied by this value to determine the actual size of the image on
      the page.  For images that have a width or height attribute, this will be applied in addition to the
      <see cref="ImageUnitMultiplier"/>.  For images without width and height, only this multiplier will be
      applied.
00244 /// </summary>
00245         public double ImageMultiplier { get; set; } = 1;
00246
00247 /// <summary>
00248 /// The margin on the right of left-aligned images and on the left of right-aligned images.
00249 /// </summary>
00250         public double ImageSideMargin { get; set; } = 10;
00251
00252 /// <summary>
00253 /// Images will be allowed to extend into the page bottom margin area by this amount before triggering
      a page break.  This should be smaller than the bottom margin, otherwise images risk being cut off by
      the page boundary.
00254 /// </summary>
00255         public double ImageMarginTolerance { get; set; } = 25;
00256
00257 /// <summary>
00258 /// A method used for syntax highlighting.  The first argument should be the source code to highlight,
      while the second parameter is the name of the language to use for the highlight.  The method should
      return a list of lists of <see cref="FormattedString"/>s, with each list of <see
      cref="FormattedString"/>s representing a line.  For each code block, if the method returns <see
      langword="null" />, no syntax highlighting is used.
00259 /// </summary>
00260         public Func<string, string, List<List<FormattedString>> SyntaxHighlighter { get; set; } =
      VectSharp.Markdown.SyntaxHighlighter.GetSyntaxHighlightedLines;
00261
00262 /// <summary>
00263 /// Bullet points used for unordered lists.  Each element of this list corresponds to the bullet for
      each level of list indentation.  If the list indentation is greater than the number of elements in
      this list, the bullet points will be reused cyclically.
00264 /// Each element of this list is a method taking two arguments:  the first is the <see
      cref="Graphics"/> object on which the bullet point should be drawn, while the second is the colour in
      which it should be painted.  The method should draw the bullet point centered around the origin.  The
      size of the bullet point will be multiplied by the current font size.
00265 /// </summary>
00266         public List<Action<Graphics, Colour» Bullets { get; } = new List<Action<Graphics, Colour»()
00267         {
00268             (graphics, colour) =>
00269             {
00270                 graphics.FillPath(new GraphicsPath().Arc(-0.5, 0, 0.25, 0, 2 * Math.PI), colour);
00271             },
00272
00273             (graphics, colour) =>
00274             {
00275                 graphics.StrokePath(new GraphicsPath().Arc(-0.5, 0, 0.25, 0, 2 * Math.PI), colour,
      0.1);
00276             },
00277
00278             (graphics, colour) =>
00279             {
00280                 graphics.StrokeRectangle(-0.75, -0.25, 0.5, 0.5, colour, 0.1);
00281             },
00282         };
00283
00284 /// <summary>
00285 /// The bullet used for unchecked task list items.
00286 /// </summary>
00287         public Graphics TaskListUncheckedBullet { get; set; } = new Func<Graphics>(() =>
00288         {
00289             Graphics tbr = new Graphics();
00290
00291             GraphicsPath checkboxPath = new GraphicsPath().MoveTo(-0.7, -0.4).LineTo(-0.3,
      -0.4).Arc(-0.3, -0.2, 0.2, 3 * Math.PI / 2, 2 * Math.PI).LineTo(-0.1, 0.2).Arc(-0.3, 0.2, 0.2, 0,
      Math.PI / 2).LineTo(-0.7, 0.4).Arc(-0.7, 0.2, 0.2, Math.PI / 2, Math.PI).LineTo(-0.9, -0.2).Arc(-0.7,
      -0.2, 0.2, Math.PI, 3 * Math.PI / 2).Close();
00292             tbr.FillPath(checkboxPath, Colour.FromRgb(240, 246, 249));
00293             tbr.StrokePath(checkboxPath, Colour.FromRgb(0, 114, 178), 0.075);
00294
00295             return tbr;
00296         })();
00297
00298 /// <summary>
00299 /// The bullet used for checked task list items.
00300 /// </summary>
00301         public Graphics TaskListCheckedBullet { get; set; } = new Func<Graphics>(() =>
00302         {
00303             Graphics tbr = new Graphics();
00304
00305             GraphicsPath checkboxPath = new GraphicsPath().MoveTo(-0.7, -0.4).LineTo(-0.3,
      -0.4).Arc(-0.3, -0.2, 0.2, 3 * Math.PI / 2, 2 * Math.PI).LineTo(-0.1, 0.2).Arc(-0.3, 0.2, 0.2, 0,
      Math.PI / 2).LineTo(-0.7, 0.4).Arc(-0.7, 0.2, 0.2, Math.PI / 2, Math.PI).LineTo(-0.9, -0.2).Arc(-0.7,
      -0.2, 0.2, Math.PI, 3 * Math.PI / 2).Close();
00306             tbr.FillPath(checkboxPath, Colour.FromRgb(240, 246, 249));
00307             tbr.StrokePath(checkboxPath, Colour.FromRgb(0, 114, 178), 0.075);
```

```
00308
00309             GraphicsPath tickpath = new GraphicsPath().MoveTo(-0.75, -0.1).LineTo(-0.5,
      0.15).LineTo(-0.1, -0.4);
00310
00311             tbr.StrokePath(new GraphicsPath().MoveTo(-0.5, 0.15).LineTo(-0.1, -0.4),
      Colour.FromRgb(240, 246, 249), 0.3, LineCaps.Round);
00312             tbr.StrokePath(tickpath, Colour.FromRgb(0, 158, 115), 0.2, LineCaps.Round);
00313
00314             return tbr;
00315         })();
00316
00317 /// <summary>
00318 /// The foreground colour for text elements.
00319 /// </summary>
00320         public Colour ForegroundColour { get; set; } = Colours.Black;
00321
00322 /// <summary>
00323 /// The background colour for the page.
00324 /// </summary>
00325         public Colour BackgroundColour { get; set; } = Colours.White;
00326
00327 /// <summary>
00328 /// The colour of the line below headers.
00329 /// </summary>
00330         public Colour HeaderLineColour { get; set; } = Colour.FromRgb(180, 180, 180);
00331
00332 /// <summary>
00333 /// The colour for thematic break lines.
00334 /// </summary>
00335         public Colour ThematicBreakLineColour { get; set; } = Colour.FromRgb(180, 180, 200);
00336
00337 /// <summary>
00338 /// The colour for hypertext links.
00339 /// </summary>
00340         public Colour LinkColour { get; set; } = Colour.FromRgb(25, 140, 191);
00341
00342 /// <summary>
00343 /// The background colour for code inlines.
00344 /// </summary>
00345         public Colour CodeInlineBackgroundColour { get; set; } = Colour.FromRgb(240, 240, 240);
00346
00347 /// <summary>
00348 /// The background colour for code blocks.
00349 /// </summary>
00350         public Colour CodeBlockBackgroundColour { get; set; } = Colour.FromRgb(240, 240, 245);
00351
00352 /// <summary>
00353 /// The colour for the bar to the left of block quotes.
00354 /// </summary>
00355         public Colour QuoteBlockBarColour { get; set; } = Colour.FromRgb(75, 152, 220);
00356
00357 /// <summary>
00358 /// The background colour for block quotes.
00359 /// </summary>
00360         public Colour QuoteBlockBackgroundColour { get; set; } = Colour.FromRgb(240, 240, 255);
00361
00362 /// <summary>
00363 /// The colour for text that has been styled as "inserted".
00364 /// </summary>
00365         public Colour InsertedColour { get; set; } = Colour.FromRgb(0, 158, 115);
00366
00367 /// <summary>
00368 /// The colour for text that has been styled as "marked".
00369 /// </summary>
00370         public Colour MarkedColour { get; set; } = Colour.FromRgb(213, 94, 0);
00371
00372 /// <summary>
00373 /// The colour for the line separating the table header row from normal rows.
00374 /// </summary>
00375         public Colour TableHeaderRowSeparatorColour { get; set; } = Colours.Black;
00376
00377 /// <summary>
00378 /// The colour for lines separating table rows from each other.
00379 /// </summary>
00380         public Colour TableRowSeparatorColour { get; set; } = Colour.FromRgb(180, 180, 180);
00381
00382 /// <summary>
00383 /// The thickness of the line separating the table header row from normal rows.
00384 /// </summary>
00385         public double TableHeaderRowSeparatorThickness { get; set; } = 2;
00386
00387 /// <summary>
00388 /// The thickness of lines separating table rows from each other.
00389 /// </summary>
00390         public double TableHeaderSeparatorThickness { get; set; } = 1;
00391
00392 /// <summary>
```

```
00393 /// Determines whether page breaks should be treated as such in the source.
00394 /// </summary>
00395         public bool AllowPageBreak { get; set; } = true;
00396 
00397 /// <summary>
00398 /// Emoji mapping that transforms emojis (e.g.  ":something:") into URLs (e.g., "emoji://something").
00399 /// </summary>
00400         public static EmojiMapping EmojiURLMapping { get; } = new EmojiMapping(new Dictionary<string,
      string>(EmojiMapping.GetDefaultEmojiShortcodeToUnicode().Select(x => new KeyValuePair<string,
      string>(x.Key, "emoji://" + x.Key.Trim(':')))), EmojiMapping.GetDefaultSmileyToEmojiShortcode());
00401 
00402 /// <summary>
00403 /// Markdown pipeline builder used to parse markdown source.
00404 /// </summary>
00405         public MarkdownPipelineBuilder MarkdownPipelineBuilder { get; set; } = new
      MarkdownPipelineBuilder().UseGridTables().UsePipeTables().UseEmphasisExtras().UseGenericAttributes().UseAutoIdentifiers
00406 
00407     }
00408 }
```

8.16 MarkdownRenderer.Tables.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2024 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using Markdig.Extensions.Tables;
00019 using Markdig.Syntax;
00020 using System;
00021 using System.Collections.Generic;
00022 using System.Linq;
00023 
00024 namespace VectSharp.Markdown
00025 {
00026     public partial class MarkdownRenderer
00027     {
00028 
00029         private void RenderTable(Table table, ref MarkdownContext context, ref Graphics graphics,
      NewPageAction newPageAction)
00030         {
00031             if (table.Count > 0)
00032             {
00033                 if (!table.IsValid())
00034                 {
00035                     table.NormalizeUsingMaxWidth();
00036                     table.NormalizeUsingHeaderRow();
00037                 }
00038 
00039                 if (table.IsValid() && table.ColumnDefinitions?.Count > 0)
00040                 {
00041                     bool isGridTable = false;
00042 
00043                     foreach (TableColumnDefinition def in table.ColumnDefinitions)
00044                     {
00045                         if (def.Width > 0)
00046                         {
00047                             isGridTable = true;
00048                             break;
00049                         }
00050                     }
00051 
00052                     if (isGridTable)
00053                     {
00054                         int maxColumns = 0;
00055 
00056                         foreach (TableRow row in table)
00057                         {
00058                             maxColumns = Math.Max(maxColumns, ((TableCell)row.Last()).ColumnIndex +
      1);
00059                         }
```

```
00060
00061                       if (table.ColumnDefinitions.Count > maxColumns)
00062                       {
00063                           table.ColumnDefinitions.RemoveRange(maxColumns,
    table.ColumnDefinitions.Count - maxColumns);
00064                       }
00065                   }
00066                   else
00067                   {
00068                       int maxColumns = 0;
00069
00070                       foreach (TableRow row in table)
00071                       {
00072                           maxColumns = Math.Max(maxColumns, row.Count);
00073                       }
00074
00075                       if (table.ColumnDefinitions.Count > maxColumns)
00076                       {
00077                           table.ColumnDefinitions.RemoveRange(maxColumns,
    table.ColumnDefinitions.Count - maxColumns);
00078                       }
00079                   }
00080
00081                   double[] columnWidths = new double[table.ColumnDefinitions.Count];
00082
00083                   if (table.ColumnDefinitions.Count == 0)
00084                   {
00085                       int columnCount = 0;
00086                       foreach (TableRow row in table)
00087                       {
00088                           columnCount = Math.Max(row.Count, columnCount);
00089                       }
00090
00091                       columnWidths = new double[columnCount];
00092                   }
00093
00094                   int missingColumns = columnWidths.Length;
00095
00096                   for (int i = 0; i < columnWidths.Length; i++)
00097                   {
00098                       columnWidths[i] = double.NaN;
00099                   }
00100
00101                   double remainingPerc = 1;
00102
00103                   for (int i = 0; i < table.ColumnDefinitions.Count; i++)
00104                   {
00105                       if (table.ColumnDefinitions[i].Width > 0)
00106                       {
00107                           missingColumns--;
00108                           remainingPerc -= table.ColumnDefinitions[i].Width / 100.0;
00109                           columnWidths[i] = table.ColumnDefinitions[i].Width / 100.0;
00110                       }
00111                   }
00112
00113                   if (missingColumns > 0)
00114                   {
00115                       remainingPerc /= missingColumns;
00116                       for (int i = 0; i < columnWidths.Length; i++)
00117                       {
00118                           if (double.IsNaN(columnWidths[i]))
00119                           {
00120                               columnWidths[i] = remainingPerc;
00121                           }
00122                       }
00123                   }
00124
00125                   double maxX = context.GetMaxX(context.Cursor.Y, context.Cursor.Y,
    this.PageSize.Width - this.Margins.Right - context.Translation.X - context.MarginBottomRight.X);
00126
00127                   for (int i = 0; i < columnWidths.Length; i++)
00128                   {
00129                       columnWidths[i] *= maxX;
00130                   }
00131
00132                   int index = 0;
00133
00134                   foreach (TableRow row in table)
00135                   {
00136                       RenderTableRow(row, columnWidths, index == table.Count - 1, ref context, ref
    graphics, newPageAction);
00137                       index++;
00138                   }
00139
00140                   context.Cursor = new Point(context.Cursor.X, context.Cursor.Y +
    SpaceAfterParagraph * context.Font.FontSize);
00141               }
```

```
00142             }
00143         }
00144
00145         private void RenderTableRow(TableRow row, double[] columnWidths, bool isLastRow, ref
      MarkdownContext context, ref Graphics graphics, NewPageAction newPageAction)
00146         {
00147             context.CurrentLine ??= new Line(0);
00148
00149             int index = 0;
00150
00151             foreach (TableCell cell in row)
00152             {
00153                 if (cell.ColumnIndex < 0)
00154                 {
00155                     cell.ColumnIndex = index;
00156                     index += cell.ColumnSpan;
00157                 }
00158                 else
00159                 {
00160                     index = cell.ColumnIndex + cell.ColumnSpan;
00161                 }
00162             }
00163
00164             double maxHeight = 0;
00165             double startX = context.Cursor.X;
00166
00167             MarkdownContext prevContext = context.Clone();
00168
00169             if (row.IsHeader)
00170             {
00171                 context.Font = new Font(this.BoldFontFamily, context.Font.FontSize);
00172             }
00173
00174
00175             foreach (TableCell cell in row)
00176             {
00177                 double cellWidth = 0;
00178
00179                 for (int i = 0; i < cell.ColumnSpan && cell.ColumnIndex + i < columnWidths.Length;
      i++)
00180                 {
00181                     cellWidth += columnWidths[cell.ColumnIndex + i];
00182                 }
00183
00184                 Page cellPage = RenderTableCell(cell, cellWidth, ref context, ref graphics,
      newPageAction);
00185
00186                 double prevMaxHeight = maxHeight;
00187                 maxHeight = Math.Max(maxHeight, cellPage.Height);
00188
00189                 context.CurrentLine.Fragments.Add(new GraphicsFragment(new Point(context.Cursor.X,
      context.Cursor.Y - cellPage.Height), cellPage.Graphics, cellPage.Height));
00190
00191                 context.Cursor = new Point(context.Cursor.X + cellWidth, context.Cursor.Y);
00192             }
00193
00194             if (this.TableVAlign == VerticalAlignment.Top)
00195             {
00196                 for (int i = 0; i < context.CurrentLine.Fragments.Count; i++)
00197                 {
00198                     context.CurrentLine.Fragments[i].Translate(0, -maxHeight +
      ((GraphicsFragment)context.CurrentLine.Fragments[i]).Ascent);
00199                 }
00200             }
00201             else if (this.TableVAlign == VerticalAlignment.Middle)
00202             {
00203                 for (int i = 0; i < context.CurrentLine.Fragments.Count; i++)
00204                 {
00205                     context.CurrentLine.Fragments[i].Translate(0, (-maxHeight +
      ((GraphicsFragment)context.CurrentLine.Fragments[i]).Ascent) * 0.5);
00206                 }
00207             }
00208
00209             context.CurrentLine.Fragments.Add(new UnderlineFragment(new Point(startX,
      context.Cursor.Y), new Point(columnWidths.Sum() + startX, context.Cursor.Y), row.IsHeader ?
      this.TableHeaderRowSeparatorColour :  this.TableRowSeparatorColour, row.IsHeader ?
      this.TableHeaderRowSeparatorThickness :  this.TableHeaderSeparatorThickness, context.Tag));
00210
00211             context.CurrentLine.Render(ref graphics, ref context, newPageAction, this.PageSize.Height
      - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y,
      context.GetMaxX(context.Cursor.Y - context.Font.Ascent, context.Cursor.Y - context.Font.Descent,
      this.PageSize.Width - this.Margins.Right - context.Translation.X - context.MarginBottomRight.X),
      this);
00212             context.CurrentLine = null;
00213
00214             context.Cursor = new Point(startX, context.Cursor.Y);
00215
```

```
00216             context.Cursor = new Point(startX, context.Cursor.Y + SpaceAfterLine *
      context.Font.FontSize + (row.IsHeader ?  this.TableHeaderRowSeparatorThickness :
      this.TableHeaderSeparatorThickness));
00217
00218             prevContext.Cursor = context.Cursor;
00219             prevContext.BottomRight = context.BottomRight;
00220             prevContext.CurrentPage = context.CurrentPage;
00221             prevContext.CurrentLine = context.CurrentLine;
00222
00223             context = prevContext;
00224         }
00225
00226         private Page RenderTableCell(TableCell cell, double cellWidth, ref MarkdownContext context,
      ref Graphics graphics, NewPageAction newPageAction)
00227         {
00228             MarkdownRenderer clonedRenderer = this.Clone();
00229             clonedRenderer.PageSize = new Size(cellWidth, double.PositiveInfinity);
00230             clonedRenderer.Margins = this.TableCellMargins;
00231
00232             MarkdownContext clonedContext = context.Clone();
00233             clonedContext.Translation = new Point(0, 0);
00234             clonedContext.Cursor = new Point(0, 0);
00235             clonedContext.BottomRight = new Point(0, 0);
00236             clonedContext.CurrentLine = null;
00237             clonedContext.CurrentPage = null;
00238             clonedContext.ForbiddenAreasLeft = new List<(double MinX, double MinY, double MaxY)>();
00239             clonedContext.ForbiddenAreasRight = new List<(double MinX, double MinY, double MaxY)>();
00240
00241             Page cellPage = clonedRenderer.RenderSubDocument(cell, ref clonedContext).Pages[0];
00242
00243             cellPage.Crop(new Point(0, 0), new Size(cellWidth, clonedContext.BottomRight.Y +
      this.TableCellMargins.Bottom));
00244
00245             return cellPage;
00246         }
00247
00248         private Document RenderSubDocument(ContainerBlock document, ref MarkdownContext context)
00249         {
00250             Document doc = new Document();
00251             Page pag = new Page(PageSize.Width, PageSize.Height) { Background = BackgroundColour };
00252             doc.Pages.Add(pag);
00253
00254             Graphics graphics = pag.Graphics;
00255
00256             graphics.Save();
00257
00258             graphics.Translate(Margins.Left, Margins.Top);
00259
00260             void newPageAction(ref MarkdownContext mdContext, ref Graphics pageGraphics)
00261             {
00262                 pageGraphics.Restore();
00263                 Page newPag = new Page(PageSize.Width, PageSize.Height) { Background =
      BackgroundColour };
00264                 doc.Pages.Add(newPag);
00265
00266                 newPag.Graphics.Save();
00267                 newPag.Graphics.Translate(mdContext.Translation);
00268                 mdContext.Cursor = new Point(0, 0);
00269                 mdContext.ForbiddenAreasLeft.Clear();
00270                 mdContext.ForbiddenAreasRight.Clear();
00271
00272                 pageGraphics = newPag.Graphics;
00273                 mdContext.CurrentPage = newPag;
00274             }
00275
00276             context.Translation = new Point(context.Translation.X + Margins.Left,
      context.Translation.Y + Margins.Top);
00277             context.CurrentPage = pag;
00278             context.ListDepth = 0;
00279
00280             int index = 0;
00281             foreach (Block block in document)
00282             {
00283                 RenderBlock(block, ref context, ref graphics, newPageAction, index > 0, index <
      document.Count - 1);
00284                 index++;
00285             }
00286
00287             context.CurrentLine?.Render(ref graphics, ref context, newPageAction, this.PageSize.Height
      - this.Margins.Bottom - context.Translation.Y - context.MarginBottomRight.Y,
      context.GetMaxX(context.Cursor.Y - context.Font.Ascent, context.Cursor.Y - context.Font.Descent,
      this.PageSize.Width - this.Margins.Right - context.Translation.X - context.MarginBottomRight.X),
      this);
00288
00289             graphics.Restore();
00290
00291             return doc;
```

```
00292         }
00293     }
00294 }
```

8.17 SyntaxHighlighting.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using Highlight;
00019 using Highlight.Engines;
00020 using Highlight.Patterns;
00021 using System;
00022 using System.Collections.Generic;
00023 using System.Linq;
00024 using System.Text;
00025 using System.Text.RegularExpressions;
00026
00027 namespace VectSharp.Markdown
00028 {
00029 /// <summary>
00030 /// Represents a string with associated formatting information.
00031 /// </summary>
00032     public struct FormattedString
00033     {
00034 /// <summary>
00035 /// The text represented by this object.
00036 /// </summary>
00037         public string Text { get; }
00038
00039 /// <summary>
00040 /// The colour of the text.
00041 /// </summary>
00042         public Colour Colour { get; }
00043
00044 /// <summary>
00045 /// Whether the text should be rendered as bold or not.
00046 /// </summary>
00047         public bool IsBold { get; }
00048
00049 /// <summary>
00050 /// Whether the text should be rendered as italic or not.
00051 /// </summary>
00052         public bool IsItalic { get; }
00053
00054 /// <summary>
00055 /// Creates a new <see cref="FormattedString"/> instance.
00056 /// </summary>
00057 /// <param name="text">The text of the object.</param>
00058 /// <param name="colour">The colour of the text.</param>
00059 /// <param name="isBold">Whether the text should be rendered as bold or not.</param>
00060 /// <param name="isItalic">Whether the text should be rendered as italic or not.</param>
00061         public FormattedString(string text, Colour colour, bool isBold, bool isItalic)
00062         {
00063             this.Text = text;
00064             this.Colour = colour;
00065             this.IsBold = isBold;
00066             this.IsItalic = isItalic;
00067         }
00068     }
00069
00070 /// <summary>
00071 /// Contains methods to perform syntax highlighting.
00072 /// </summary>
00073     public static class SyntaxHighlighter
00074     {
00075         private static Dictionary<string, string[]> LanguageAliases = new Dictionary<string,
      string[]>()
00076         {
```

```
00077             { "ASPX", new string[] { "ASP.NET", "aspx", "aspx-vb" } },
00078             { "C", new string[] { } },
00079             { "C++", new string[] { "cpp" } },
00080             { "C#", new string[] { "csharp" } },
00081             { "COBOL", new string[] { } },
00082             { "Eiffel", new string[] { } },
00083             { "Fortran", new string[] { } },
00084             { "Haskell", new string[] { } },
00085             { "HTML", new string[] { "xhtml" } },
00086             { "Java", new string[] { } },
00087             { "JavaScript", new string[] { "js", "node" } },
00088             { "Mercury", new string[] { } },
00089             { "MSIL", new string[] { } },
00090             { "Pascal", new string[] { "delphi", "objectpascal" } },
00091             { "Perl", new string[] { "cperl" } },
00092             { "PHP", new string[] { "inc" } },
00093             { "Python", new string[] { "python3", "rusthon" } },
00094             { "Ruby", new string[] { "jruby", "macruby", "rake", "rb", "rbx" } },
00095             { "SQL", new string[] { } },
00096             { "Visual Basic", new string[] { "vba", "vb6", "visual basic 6", "visual basic for
      applications" } },
00097             { "VBScript", new string[] { } },
00098             { "VB.NET", new string[] { "Visual Basic .NET", "vbnet", "vb .net" } },
00099             { "XML", new string[] { "rss", "xsd", "wsdl" } },
00100         };
00101
00102         private static string GetLanguage(string language)
00103         {
00104             foreach (KeyValuePair<string, string[]> element in LanguageAliases)
00105             {
00106                 if (element.Key.Equals(language, StringComparison.OrdinalIgnoreCase))
00107                 {
00108                     return element.Key;
00109                 }
00110
00111                 foreach (string alias in element.Value)
00112                 {
00113                     if (alias.Equals(language, StringComparison.OrdinalIgnoreCase))
00114                     {
00115                         return element.Key;
00116                     }
00117                 }
00118             }
00119
00120             return null;
00121         }
00122
00123 /// <summary>
00124 /// Performs syntax highlighting for a specified language on some source code.
00125 /// </summary>
00126 /// <param name="sourceCode">The source code to be highlighted.</param>
00127 /// <param name="language">The name of the language to use for the highlighting.</param>
00128 /// <returns>A list of lists of <see cref="FormattedString"/>s.  Each list of <see
      cref="FormattedString"/>s represents a line.</returns>
00129         public static List<List<FormattedString» GetSyntaxHighlightedLines(string sourceCode, string
      language)
00130         {
00131             language = GetLanguage(language);
00132
00133             if (string.IsNullOrEmpty(language))
00134             {
00135                 return null;
00136             }
00137
00138             HighlighterEngine engine = new HighlighterEngine();
00139
00140             Highlighter highlighter = new Highlighter(engine);
00141
00142             highlighter.Highlight(language, sourceCode);
00143
00144             List<List<FormattedString» tbr = new List<List<FormattedString»();
00145
00146             List<FormattedString> currentLine = new List<FormattedString>();
00147
00148             for (int i = 0; i < engine.HighlightedSpans.Count; i++)
00149             {
00150                 string[] split = engine.HighlightedSpans[i].Text.Replace("\r", "").Split('\n');
00151
00152                 for (int j = 0; j < split.Length; j++)
00153                 {
00154                     currentLine.Add(new FormattedString(split[j], engine.HighlightedSpans[i].Colour,
      engine.HighlightedSpans[i].IsBold, engine.HighlightedSpans[i].IsItalic));
00155
00156                     if (j < split.Length - 1)
00157                     {
00158                         tbr.Add(currentLine);
00159                         currentLine = new List<FormattedString>();
```

```
00160                     }
00161                 }
00162             }
00163
00164             tbr.Add(currentLine);
00165
00166             return tbr;
00167         }
00168     }
00169
00170     internal class HighlighterEngine :  Engine, IEngine
00171     {
00172         private const RegexOptions DefaultRegexOptions = RegexOptions.ExplicitCapture |
      RegexOptions.IgnorePatternWhitespace;
00173
00174         public List<FormattedString> HighlightedSpans { get; } = new List<FormattedString>();
00175
00176         protected override string ProcessBlockPatternMatch(Definition definition, BlockPattern
      pattern, Match match)
00177         {
00178             if (!string.IsNullOrEmpty(pattern.RawRegex))
00179             {
00180                 Match reMatch = Regex.Match(match.Value, pattern.RawRegex);
00181
00182                 if (reMatch.Groups.Count > 1)
00183                 {
00184                     HighlightedSpans.Add(new FormattedString(match.Value.Substring(0,
      reMatch.Groups[1].Index), Colours.Black, false, false));
00185                     HighlightedSpans.Add(new FormattedString(reMatch.Groups[1].Value,
      pattern.Style.Colors.ForeColor, pattern.Style.Font.IsBold, pattern.Style.Font.IsItalic));
00186                     HighlightedSpans.Add(new
      FormattedString(match.Value.Substring(reMatch.Groups[1].Index + reMatch.Groups[1].Length),
      Colours.Black, false, false));
00187                 }
00188                 else
00189                 {
00190                     HighlightedSpans.Add(new FormattedString(match.Value,
      pattern.Style.Colors.ForeColor, pattern.Style.Font.IsBold, pattern.Style.Font.IsItalic));
00191                 }
00192             }
00193             else
00194             {
00195                 HighlightedSpans.Add(new FormattedString(match.Value, pattern.Style.Colors.ForeColor,
      pattern.Style.Font.IsBold, pattern.Style.Font.IsItalic));
00196             }
00197             return match.Value;
00198         }
00199
00200         protected override string ProcessMarkupPatternMatch(Definition definition, MarkupPattern
      pattern, Match match)
00201         {
00202             HighlightedSpans.Add(new FormattedString(match.Value, pattern.Style.Colors.ForeColor,
      pattern.Style.Font.IsBold, pattern.Style.Font.IsItalic));
00203             return match.Value;
00204         }
00205
00206         protected override string ProcessWordPatternMatch(Definition definition, WordPattern pattern,
      Match match)
00207         {
00208             HighlightedSpans.Add(new FormattedString(match.Value, pattern.Style.Colors.ForeColor,
      pattern.Style.Font.IsBold, pattern.Style.Font.IsItalic));
00209             return match.Value;
00210         }
00211
00212         string IEngine.Highlight(Definition definition, string input)
00213         {
00214             if (definition == null)
00215             {
00216                 throw new ArgumentNullException("definition");
00217             }
00218
00219             var output = PreHighlight(definition, input);
00220             output = HighlightUsingRegex(definition, output);
00221             output = PostHighlight(definition, output);
00222
00223             return output;
00224         }
00225
00226         private string HighlightUsingRegex(Definition definition, string input)
00227         {
00228             var regexOptions = GetRegexOptions(definition);
00229             var evaluator = GetMatchEvaluator(definition);
00230             var regexPattern = definition.GetRegexPattern();
00231
00232             int currentIndex = 0;
00233
00234             foreach (Match match in Regex.Matches(input, regexPattern))
```

```
00235             {
00236                 if (match.Index > currentIndex)
00237                 {
00238                     this.HighlightedSpans.Add(new FormattedString(input.Substring(currentIndex,
      match.Index - currentIndex), Colours.Black, false, false));
00239                 }
00240 
00241                 currentIndex = match.Index + match.Length;
00242 
00243                 evaluator(match);
00244             }
00245 
00246             if (currentIndex < input.Length)
00247             {
00248                 this.HighlightedSpans.Add(new FormattedString(input.Substring(currentIndex,
      input.Length - currentIndex), Colours.Black, false, false));
00249             }
00250 
00251             return null;
00252         }
00253 
00254         private RegexOptions GetRegexOptions(Definition definition)
00255         {
00256             if (definition.CaseSensitive)
00257             {
00258                 return DefaultRegexOptions | RegexOptions.IgnoreCase;
00259             }
00260 
00261             return DefaultRegexOptions;
00262         }
00263 
00264         private string ElementMatchHandler(Definition definition, Match match)
00265         {
00266             if (definition == null)
00267             {
00268                 throw new ArgumentNullException("definition");
00269             }
00270             if (match == null)
00271             {
00272                 throw new ArgumentNullException("match");
00273             }
00274 
00275             var pattern = definition.Patterns.First(x => match.Groups[x.Key].Success).Value;
00276             if (pattern != null)
00277             {
00278                 if (pattern is BlockPattern)
00279                 {
00280                     return ProcessBlockPatternMatch(definition, (BlockPattern)pattern, match);
00281                 }
00282                 if (pattern is MarkupPattern)
00283                 {
00284                     return ProcessMarkupPatternMatch(definition, (MarkupPattern)pattern, match);
00285                 }
00286                 if (pattern is WordPattern)
00287                 {
00288                     return ProcessWordPatternMatch(definition, (WordPattern)pattern, match);
00289                 }
00290             }
00291 
00292             return match.Value;
00293         }
00294 
00295         private MatchEvaluator GetMatchEvaluator(Definition definition)
00296         {
00297             return match => ElementMatchHandler(definition, match);
00298         }
00299     }
00300 }
```

8.18 UnicodeEmojiParser.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2024 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
```

```
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using Markdig;
00019 using Markdig.Extensions.Emoji;
00020 using Markdig.Helpers;
00021 using Markdig.Parsers;
00022 using Markdig.Parsers.Inlines;
00023 using Markdig.Renderers;
00024 using System.Globalization;
00025 using System.Linq;
00026
00027 namespace VectSharp.Markdown
00028 {
00029 /// <summary>
00030 /// Contains extension methods for the <see cref="MarkdownPipelineBuilder"/> class.
00031 /// </summary>
00032     public static class MarkdownEmojiExtension
00033     {
00034 /// <summary>
00035 /// Uses the extension enabling unicode emoji resolution.
00036 /// </summary>
00037 /// <param name="pipeline">The pipeline.</param>
00038 /// <returns>The modified pipeline.</returns>
00039         public static MarkdownPipelineBuilder UseUnicodeEmoji(this MarkdownPipelineBuilder pipeline)
00040         {
00041             if (!pipeline.Extensions.Contains<UnicodeEmojiExtension>())
00042             {
00043                 pipeline.Extensions.Add(new UnicodeEmojiExtension());
00044             }
00045
00046             return pipeline;
00047         }
00048     }
00049
00050     internal class UnicodeEmojiExtension :  IMarkdownExtension
00051     {
00052         public void Setup(MarkdownPipelineBuilder pipeline)
00053         {
00054             if (!pipeline.InlineParsers.Contains<UnicodeEmojiParser>())
00055             {
00056                 pipeline.InlineParsers.InsertAfter<CodeInlineParser>(new UnicodeEmojiParser());
00057             }
00058         }
00059
00060         public void Setup(MarkdownPipeline pipeline, IMarkdownRenderer renderer)
00061         {
00062
00063         }
00064     }
00065
00066     internal class UnicodeEmojiParser :  InlineParser
00067     {
00068         public UnicodeEmojiParser()
00069         {
00070             OpeningCharacters = Enumerable.Range(0xD800, 0xDBFF - 0xD800 + 1).Select(x =>
    (char)x).ToArray();
00071         }
00072
00073         public override bool Match(InlineProcessor processor, ref StringSlice slice)
00074         {
00075             if (char.IsHighSurrogate(slice.CurrentChar))
00076             {
00077                 StringInfo info = new StringInfo(slice.Text.Substring(slice.Start));
00078                 string emoji = info.SubstringByTextElements(0, 1);
00079
00080                 processor.Inline = new EmojiInline("unicode://" + emoji)
00081                 {
00082                     Span = {
00083                         Start = processor.GetSourcePosition(slice.Start, out int line, out int column)
00084                     },
00085                     Line = line,
00086                     Column = column,
00087                 };
00088                 processor.Inline.Span.End = processor.Inline.Span.Start + emoji.Length - 1;
00089
00090                 slice.Start += emoji.Length;
00091
00092                 return true;
00093             }
00094             else
00095             {
00096                 return false;
00097             }
00098         }
```

```
00099     }
00100 }
```

8.19 Word.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Text;
00021
00022 namespace VectSharp.Markdown
00023 {
00024     internal class Word
00025     {
00026         public char PrecedingWhitespace { get; }
00027         public int WhitespaceCount { get; }
00028         public string Text { get; }
00029
00030         public Font.DetailedFontMetrics Metrics { get; }
00031
00032         private Word(char precedingWhitespace, int whitespaceCount, string text, Font font)
00033         {
00034             this.PrecedingWhitespace = precedingWhitespace;
00035             this.WhitespaceCount = whitespaceCount;
00036             this.Text = text;
00037
00038             if (!string.IsNullOrEmpty(text))
00039             {
00040                 this.Metrics = font.MeasureTextAdvanced(text);
00041             }
00042             else
00043             {
00044                 this.Metrics = font.MeasureTextAdvanced(" ");
00045             }
00046         }
00047
00048         private static Dictionary<FontFamily, double> SpaceWidths { get; } = new
      Dictionary<FontFamily, double>();
00049
00050         private static IEnumerable<Word> SplitWord(Word word, Font font, double maxWidth)
00051         {
00052             if (!SpaceWidths.TryGetValue(font.FontFamily, out double spaceWidth))
00053             {
00054                 spaceWidth = font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(' ') / 1000.0;
00055                 SpaceWidths[font.FontFamily] = spaceWidth;
00056             }
00057
00058             spaceWidth *= font.FontSize;
00059
00060             while (!string.IsNullOrEmpty(word.Text) && word.Metrics.Width +
      word.Metrics.LeftSideBearing + word.Metrics.RightSideBearing + spaceWidth * word.WhitespaceCount *
      (word.PrecedingWhitespace == '\t' ?  4 :  1) > maxWidth)
00061             {
00062                 int minIndex = 1;
00063                 int maxIndex = word.Text.Length;
00064
00065                 while (maxIndex - minIndex > 1)
00066                 {
00067                     int index = (minIndex + maxIndex) / 2;
00068
00069                     double width = font.MeasureText(word.Text.Substring(0, index)).Width;
00070
00071                     if (width > maxWidth)
00072                     {
00073                         maxIndex = index;
00074                     }
00075                     else if (width < maxWidth)
```

```
00076                     {
00077                         minIndex = index;
00078                     }
00079                     else
00080                     {
00081                         minIndex = index;
00082                         maxIndex = index;
00083                     }
00084                 }
00085 
00086                 Word newWord = new Word(word.PrecedingWhitespace, word.WhitespaceCount,
      word.Text.Substring(0, minIndex), font);
00087                 yield return newWord;
00088 
00089                 word = new Word('\0', 0, word.Text.Substring(minIndex), font);
00090             }
00091 
00092             yield return word;
00093         }
00094 
00095         public static IEnumerable<Word> GetWords(string text, Font font, double maxWidth)
00096         {
00097             if (!SpaceWidths.TryGetValue(font.FontFamily, out double spaceWidth))
00098             {
00099                 spaceWidth = font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(' ') / 1000.0;
00100                 SpaceWidths[font.FontFamily] = spaceWidth;
00101             }
00102 
00103             spaceWidth *= font.FontSize;
00104 
00105             foreach (Word w in GetWordsInternal(text, font))
00106             {
00107                 if (w.Metrics.Width + w.Metrics.LeftSideBearing + w.Metrics.RightSideBearing +
      spaceWidth * w.WhitespaceCount * (w.PrecedingWhitespace == '\t' ?  4 :  1) > maxWidth)
00108                 {
00109                     foreach (Word w2 in SplitWord(w, font, maxWidth))
00110                     {
00111                         yield return w2;
00112                     }
00113                 }
00114                 else
00115                 {
00116                     yield return w;
00117                 }
00118             }
00119         }
00120 
00121         private static IEnumerable<Word> GetWordsInternal(string text, Font font)
00122         {
00123             StringBuilder currWord = new StringBuilder();
00124 
00125             char currWhitespace = '\0';
00126             int whitespaceCount = 0;
00127 
00128             for (int i = 0; i < text.Length; i++)
00129             {
00130                 if (char.IsWhiteSpace(text[i]))
00131                 {
00132                     if (currWord.Length > 0)
00133                     {
00134                         yield return new Word(currWhitespace, whitespaceCount, currWord.ToString(),
      font);
00135 
00136                         currWord.Clear();
00137                         currWhitespace = text[i];
00138                         whitespaceCount = 1;
00139                     }
00140                     else if (currWhitespace == text[i])
00141                     {
00142                         whitespaceCount++;
00143                     }
00144                     else
00145                     {
00146                         yield return new Word(currWhitespace, whitespaceCount, null, font);
00147 
00148                         currWhitespace = text[i];
00149                         whitespaceCount = 1;
00150                     }
00151                 }
00152                 else
00153                 {
00154                     currWord.Append(text[i]);
00155                 }
00156             }
00157 
00158             if (currWord.Length > 0)
00159             {
```

```
00160                 yield return new Word(currWhitespace, whitespaceCount, currWord.ToString(), font);
00161
00162                 currWord.Clear();
00163             }
00164             else if (whitespaceCount > 0)
00165             {
00166                 yield return new Word(currWhitespace, whitespaceCount, null, font);
00167             }
00168         }
00169     }
00170 }
```

8.20 ImageCache.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using Avalonia.Controls.ApplicationLifetimes;
00019 using System;
00020 using System.Collections.Generic;
00021 using System.IO;
00022
00023 namespace VectSharp.MarkdownCanvas
00024 {
00025     internal static class ImageCache
00026     {
00027         private const string imageCacheId = "bb5a724e-93f7-431d-87c3-53f31d8da16e";
00028
00029         private static string imageCacheFolder;
00030
00031         private static Dictionary<string, string> imageCache;
00032
00033         static ImageCache()
00034         {
00035             imageCacheFolder = Path.Combine(Path.GetTempPath(), imageCacheId);
00036             Directory.CreateDirectory(imageCacheFolder);
00037             imageCache = new Dictionary<string, string>();
00038         }
00039
00040         private static bool exitHandlerSet = false;
00041         public static void SetExitEventHandler()
00042         {
00043             if (!exitHandlerSet)
00044             {
00045                 if (Avalonia.Application.Current.ApplicationLifetime is IControlledApplicationLifetime
      lifetime)
00046                 {
00047                     lifetime.Exit += (s, e) =>
00048                     {
00049                         try
00050                         {
00051                             Directory.Delete(imageCacheFolder, true);
00052                         }
00053                         catch
00054                         {
00055                             try
00056                             {
00057                                 foreach (string sr in Directory.GetFiles(imageCacheFolder, "*.*"))
00058                                 {
00059                                     try
00060                                     {
00061                                         File.Delete(sr);
00062                                     }
00063                                     catch { }
00064                                 }
00065                             }
00066                             catch { }
00067                         }
00068                     };
```

```
00069                 }
00070                 exitHandlerSet = true;
00071             }
00072         }
00073 
00074         public static (string, bool) ImageUriResolver(string imageUri, string baseUriString)
00075         {
00076             if (!imageCache.TryGetValue(baseUriString + "|||" + imageUri, out string cachedImage))
00077             {
00078                 (string imagePath, bool wasDownloaded) =
      VectSharp.Markdown.HTTPUtils.ResolveImageURI(imageUri, baseUriString);
00079 
00080                 if (!string.IsNullOrEmpty(imagePath) && File.Exists(imagePath))
00081                 {
00082                     string id = Guid.NewGuid().ToString();
00083 
00084                     cachedImage = Path.Combine(imageCacheFolder, id + Path.GetExtension(imagePath));
00085 
00086                     if (wasDownloaded)
00087                     {
00088                         if (!Directory.Exists(imageCacheFolder))
00089                         {
00090                             Directory.CreateDirectory(imageCacheFolder);
00091                         }
00092 
00093                         File.Move(imagePath, cachedImage);
00094                         Directory.Delete(Path.GetDirectoryName(imagePath));
00095                     }
00096                     else
00097                     {
00098                         File.Copy(imagePath, cachedImage);
00099                     }
00100 
00101                     imageCache[baseUriString + "|||" + imageUri] = cachedImage;
00102                 }
00103                 else
00104                 {
00105                     cachedImage = null;
00106                 }
00107             }
00108 
00109             return (cachedImage, false);
00110         }
00111     }
00112 }
```

8.21 MarkdownCanvas.axaml.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using Avalonia;
00019 using Avalonia.Controls;
00020 using Avalonia.Controls.ApplicationLifetimes;
00021 using Avalonia.Markup.Xaml;
00022 using Avalonia.Media;
00023 using Avalonia.Threading;
00024 using Markdig;
00025 using Markdig.Syntax;
00026 using System;
00027 using System.Collections.Generic;
00028 using VectSharp.Canvas;
00029 using VectSharp.Markdown;
00030 
00031 namespace VectSharp.MarkdownCanvas
00032 {
00033 /// <summary>
00034 /// A control to display a Markdown document in an Avalonia application.
00035 /// </summary>
```

```
00036     public partial class MarkdownCanvasControl :  UserControl
00037     {
00038 /// <summary>
00039 /// Defines the <see cref="MaxRenderWidth"/> property.
00040 /// </summary>
00041         public static readonly StyledProperty<double> MaxRenderWidthProperty =
      AvaloniaProperty.Register<MarkdownCanvasControl, double>(nameof(MaxRenderWidth),
      double.PositiveInfinity);
00042
00043 /// <summary>
00044 /// The maximum width for the rendered document.  This will be used even if the control's client area
      is larger than this (the alignment of the document within the controll will depend on the control's
      <see cref="ContentControl.HorizontalContentAlignment"/>).
00045 /// </summary>
00046         public double MaxRenderWidth
00047         {
00048             get { return GetValue(MaxRenderWidthProperty); }
00049             set { SetValue(MaxRenderWidthProperty, value); }
00050         }
00051
00052 /// <summary>
00053 /// Defines the <see cref="MinRenderWidth"/> property.
00054 /// </summary>
00055         public static readonly StyledProperty<double> MinRenderWidthProperty =
      AvaloniaProperty.Register<MarkdownCanvasControl, double>(nameof(MinRenderWidth), 200);
00056
00057 /// <summary>
00058 /// The minimum width for the rendered document.  If the control's client area is smaller than this,
      the horizontal scroll bar will be activated.
00059 /// </summary>
00060         public double MinRenderWidth
00061         {
00062             get { return GetValue(MinRenderWidthProperty); }
00063             set { SetValue(MinRenderWidthProperty, value); }
00064         }
00065
00066 /// <summary>
00067 /// Defines the <see cref="MinVariation"/> property.
00068 /// </summary>
00069         public static readonly StyledProperty<double> MinVariationProperty =
      AvaloniaProperty.Register<MarkdownCanvasControl, double>(nameof(MinVariation), 10);
00070
00071 /// <summary>
00072 /// The minimum width variation that triggers a document reflow.  If the control is resized, but the
      width changes by less than this amount, the document is not re-drawn.
00073 /// </summary>
00074         public double MinVariation
00075         {
00076             get { return GetValue(MinVariationProperty); }
00077             set { SetValue(MinVariationProperty, value); }
00078         }
00079
00080 /// <summary>
00081 /// Defines the <see cref="DocumentSource"/> property.
00082 /// </summary>
00083         public static readonly StyledProperty<string> DocumentSourceProperty =
      AvaloniaProperty.Register<MarkdownCanvasControl, string>(nameof(DocumentSource));
00084
00085 /// <summary>
00086 /// Sets the currently displayed document from Markdown source.
00087 /// </summary>
00088         public string DocumentSource
00089         {
00090             set { SetValue(DocumentSourceProperty, value); }
00091         }
00092
00093 /// <summary>
00094 /// Defines the <see cref="Document"/> property.
00095 /// </summary>
00096         public static readonly StyledProperty<MarkdownDocument> DocumentProperty =
      AvaloniaProperty.Register<MarkdownCanvasControl, MarkdownDocument>(nameof(Document));
00097
00098 /// <summary>
00099 /// Gets or sets the currently displayed <see cref="MarkdownDocument"/>.
00100 /// </summary>
00101         public MarkdownDocument Document
00102         {
00103             get { return GetValue(DocumentProperty); }
00104             set { SetValue(DocumentProperty, value); }
00105         }
00106
00107 /// <summary>
00108 /// Defines the <see cref="TextConversionOption"/> property.
00109 /// </summary>
00110         public static readonly StyledProperty<AvaloniaContextInterpreter.TextOptions>
      TextConversionOptionsProperty = AvaloniaProperty.Register<MarkdownCanvasControl,
      AvaloniaContextInterpreter.TextOptions>(nameof(TextConversionOption),
```

```
    AvaloniaContextInterpreter.TextOptions.ConvertIfNecessary);
00111
00112 /// <summary>
00113 /// Gets or sets the value that determines whether text items should be converted into paths when
    drawing.
00114 /// Setting this to <see cref="AvaloniaContextInterpreter.TextOptions.NeverConvert"/> will improve
    performance if you are using custom fonts, but may cause unexpected results unless the font families
    being used are of type <see cref="ResourceFontFamily"/>.
00115 /// </summary>
00116         public AvaloniaContextInterpreter.TextOptions TextConversionOption
00117         {
00118             get { return GetValue(TextConversionOptionsProperty); }
00119             set { SetValue(TextConversionOptionsProperty, value); }
00120         }
00121
00122 /// <summary>
00123 /// The <see cref="MarkdownRenderer"/> used to render the <see cref="Document"/>.  You can use the
    properties of this object to customise the rendering.  Note that setting the <see
    cref="Avalonia.Controls.Primitives.TemplatedControl.FontSize"/> of the <see
    cref="MarkdownCanvasControl"/> will propagate to the <see cref="Renderer"/>'s <see
    cref="MarkdownRenderer.BaseFontSize"/>.
00124 /// </summary>
00125         public MarkdownRenderer Renderer { get; }
00126
00127         private double lastRenderedWidth = double.NaN;
00128
00129         private bool initialized = false;
00130
00131 /// <summary>
00132 /// Initialises a new <see cref="MarkdownCanvasControl"/>.
00133 /// </summary>
00134         public MarkdownCanvasControl()
00135         {
00136             InitializeComponent();
00137             this.Renderer = new MarkdownRenderer() { BaseFontSize = this.FontSize, Margins = new
    Margins(10, 10, 10, 10), ImageUriResolver = ImageCache.ImageUriResolver };
00138             this.initialized = true;
00139             ImageCache.SetExitEventHandler();
00140         }
00141
00142 /// <inheritdoc/>
00143         protected override void OnPropertyChanged(AvaloniaPropertyChangedEventArgs change)
00144         {
00145             if (change.Property != ContentProperty || !initialized)
00146             {
00147                 base.OnPropertyChanged(change);
00148
00149                 if (this.initialized)
00150                 {
00151                     if (change.Property == MarkdownCanvasControl.BoundsProperty || change.Property ==
    MarkdownCanvasControl.MinRenderWidthProperty || change.Property ==
    MarkdownCanvasControl.MaxRenderWidthProperty || change.Property ==
    MarkdownCanvasControl.MinVariationProperty)
00152                     {
00153                         Render();
00154                     }
00155                     else if (change.Property == MarkdownCanvasControl.DocumentProperty)
00156                     {
00157                         forcedRerender = true;
00158                         Render();
00159                     }
00160                     else if (change.Property == MarkdownCanvasControl.FontSizeProperty)
00161                     {
00162                         this.Renderer.BaseFontSize = this.FontSize;
00163                         forcedRerender = true;
00164                         Render();
00165                     }
00166                     else if (change.Property == MarkdownCanvasControl.DocumentSourceProperty)
00167                     {
00168                         MarkdownDocument document =
    Markdig.Markdown.Parse(change.GetNewValue<string>(), Renderer.MarkdownPipelineBuilder.Build());
00169                         this.Document = document;
00170                     }
00171                     else if (change.Property ==
    MarkdownCanvasControl.VerticalContentAlignmentProperty)
00172                     {
00173                         this.FindControl<ScrollViewer>("ScrollViewer").VerticalContentAlignment =
    this.VerticalContentAlignment;
00174                     }
00175                     else if (change.Property ==
    MarkdownCanvasControl.HorizontalContentAlignmentProperty)
00176                     {
00177                         this.FindControl<ScrollViewer>("ScrollViewer").HorizontalContentAlignment =
    this.HorizontalContentAlignment;
00178                     }
00179                 }
00180             }
```

```
00181             else
00182             {
00183                 initialized = false;
00184                 this.Content = change.OldValue;
00185                 initialized = true;
00186             }
00187         }
00188 
00189         private bool forcedRerender = false;
00190 
00191         private void Render()
00192         {
00193             if (Document != null)
00194             {
00195                 double width = Math.Min(MaxRenderWidth, Math.Max(MinRenderWidth, this.Bounds.Width -
      MinVariation - 13));
00196 
00197                 if (forcedRerender || double.IsNaN(lastRenderedWidth) || width != lastRenderedWidth &&
      width < lastRenderedWidth - MinVariation || width > lastRenderedWidth + MinVariation)
00198                 {
00199                     Page pag;
00200                     Dictionary<string, string> linkDestinations;
00201 
00202                     try
00203                     {
00204                         pag = Renderer.RenderSinglePage(this.Document, width, out linkDestinations,
      out _);
00205                     }
00206                     catch
00207                     {
00208                         pag = new Page(width, 0);
00209                         linkDestinations = new Dictionary<string, string>();
00210                     }
00211 
00212                     Dictionary<string, Delegate> taggedActions = new Dictionary<string, Delegate>();
00213                     Dictionary<string, Avalonia.Point> linkDestinationPoints = new Dictionary<string,
      Avalonia.Point>();
00214 
00215                     foreach (KeyValuePair<string, string> linkDestination in linkDestinations)
00216                     {
00217                         string url = linkDestination.Value;
00218 
00219                         taggedActions.Add(linkDestination.Key, (Func<RenderAction,
      IEnumerable<RenderAction>>)(act =>
00220                         {
00221                             act.PointerEntered += (s, e) =>
00222                             {
00223                                 act.Parent.Cursor = new
      Avalonia.Input.Cursor(Avalonia.Input.StandardCursorType.Hand);
00224                             };
00225 
00226                             act.PointerExited += (s, e) =>
00227                             {
00228                                 act.Parent.Cursor = new
      Avalonia.Input.Cursor(Avalonia.Input.StandardCursorType.Arrow);
00229                             };
00230 
00231                             act.PointerPressed += (s, e) =>
00232                             {
00233                                 if (url.StartsWith("#"))
00234                                 {
00235                                     if (linkDestinationPoints.TryGetValue(url.Substring(1), out
      Avalonia.Point target))
00236                                     {
00237                                         ScrollViewer scrollViewer =
      this.FindControl<ScrollViewer>("ScrollViewer");
00238 
00239                                         scrollViewer.Offset = new
      Vector(Math.Max(Math.Min(scrollViewer.Offset.X, target.X), target.X - scrollViewer.Viewport.Width),
      target.Y);
00240                                     }
00241                                 }
00242                                 else
00243                                 {
00244                                     System.Diagnostics.Process.Start(new
      System.Diagnostics.ProcessStartInfo() { FileName = url, UseShellExecute = true });
00245                                 }
00246                             };
00247 
00248                             if (act.ActionType == RenderAction.ActionTypes.Path)
00249                             {
00250                                 linkDestinationPoints[linkDestination.Key] =
      act.Geometry.Bounds.TopLeft.Transform(act.Transform);
00251                             }
00252                             else if (act.ActionType == RenderAction.ActionTypes.Text)
00253                             {
00254                                 linkDestinationPoints[linkDestination.Key] = new Avalonia.Point(0,
```

```
0).Transform(act.Transform);
00255                             }
00256                             else if (act.ActionType == RenderAction.ActionTypes.RasterImage)
00257                             {
00258                                 linkDestinationPoints[linkDestination.Key] =
act.ImageDestination.Value.TopLeft.Transform(act.Transform);
00259                             }
00260 
00261                             return new RenderAction[] { act };
00262                         }));
00263 
00264                         if (url.StartsWith("#"))
00265                         {
00266                             if (!taggedActions.ContainsKey(url.Substring(1)))
00267                             {
00268                                 taggedActions.Add(url.Substring(1), (Func<RenderAction,
IEnumerable<RenderAction>>)(act =>
00269                                 {
00270                                     if (act.ActionType == RenderAction.ActionTypes.Path)
00271                                     {
00272                                         linkDestinationPoints[url.Substring(1)] =
act.Geometry.Bounds.TopLeft.Transform(act.Transform);
00273                                     }
00274                                     else if (act.ActionType == RenderAction.ActionTypes.Text)
00275                                     {
00276                                         linkDestinationPoints[url.Substring(1)] = new
Avalonia.Point(0, 0).Transform(act.Transform);
00277                                     }
00278                                     else if (act.ActionType == RenderAction.ActionTypes.RasterImage)
00279                                     {
00280                                         linkDestinationPoints[url.Substring(1)] =
act.ImageDestination.Value.TopLeft.Transform(act.Transform);
00281                                     }
00282 
00283                                     return new RenderAction[] { act };
00284                                 }));
00285                             }
00286                         }
00287                     }
00288 
00289                     Control can = pag.PaintToCanvas(false, taggedActions, false,
this.TextConversionOption);
00290 
00291                     this.FindControl<ScrollViewer>("ScrollViewer").Content = can;
00292                     this.FindControl<ScrollViewer>("ScrollViewer").Padding = new Thickness(0, 0, 0,
0);
00293                     lastRenderedWidth = width;
00294                     forcedRerender = false;
00295                 }
00296                 else
00297                 {
00298                     this.FindControl<ScrollViewer>("ScrollViewer").Padding = new Thickness(0, 0, width
- lastRenderedWidth, 0);
00299                 }
00300             }
00301         }
00302 
00303         private void InitializeComponent()
00304         {
00305             AvaloniaXamlLoader.Load(this);
00306         }
00307     }
00308 }
```

8.22 ImageUriParser.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
```

```
00018 using SixLabors.ImageSharp;
00019 using System;
00020 using System.IO;
00021 using System.Linq;
00022 using System.Runtime.InteropServices;
00023
00024 namespace VectSharp.ImageSharpUtils
00025 {
00026 /// <summary>
00027 /// Provides a method to parse an image URI into a page.
00028 /// </summary>
00029     public static class ImageURIParser
00030     {
00031 /// <summary>
00032 /// Parses an image URI into a page.  This is intended to replace the default image URI interpreter in
      <c>VectSharp.SVG.Parser.ParseImageURI</c>.  To do this, use something like:
00033 /// <code>VectSharp.SVG.Parser.ParseImageURI =
      VectSharp.ImageSharpUtils.ImageURIParser.Parser(VectSharp.SVG.Parser.ParseSVGURI);</code>
00034 /// </summary>
00035 /// <param name="parseSVG">A function to parse an SVG image uri into a page.  You should pass
      <c>VectSharp.SVG.Parser.ParseSVGURI</c> as this argument.</param>
00036 /// <returns>A function to parse an image URI into a page.</returns>
00037         public static Func<string, bool, Page> Parser(Func<string, bool, Page> parseSVG)
00038         {
00039             return (string uri, bool interpolate) =>
00040             {
00041                 if (uri.StartsWith("data:"))
00042                 {
00043                     string mimeType = uri.Substring(uri.IndexOf(":") + 1, uri.IndexOf(";") -
      uri.IndexOf(":") - 1);
00044
00045                     string type = uri.Substring(uri.IndexOf(";") + 1, uri.IndexOf(",") -
      uri.IndexOf(";") - 1);
00046
00047                     if (mimeType != "image/svg+xml")
00048                     {
00049                         int offset = uri.IndexOf(",") + 1;
00050
00051                         byte[] parsed;
00052
00053                         string substring = uri.Substring(offset);
00054
00055                         switch (type)
00056                         {
00057                             case "base64":
00058                                 parsed = Convert.FromBase64String(uri.Substring(offset));
00059                                 break;
00060                             case "":
00061                                 parsed = (from el in
      System.Web.HttpUtility.UrlDecode(uri.Substring(offset)) select (byte)el).ToArray();
00062                                 break;
00063                             default:
00064                                 throw new InvalidDataException("Unknown data stream type!");
00065                         }
00066
00067                         GCHandle handle = GCHandle.Alloc(parsed, GCHandleType.Pinned);
00068
00069                         RasterImageStream img = new RasterImageStream(handle.AddrOfPinnedObject(),
      parsed.Length, interpolate:  interpolate);
00070
00071                         handle.Free();
00072
00073                         Page pag = new Page(img.Width, img.Height);
00074
00075                         pag.Graphics.DrawRasterImage(0, 0, img);
00076
00077                         return pag;
00078                     }
00079                     else
00080                     {
00081                         return parseSVG(uri, interpolate);
00082                     }
00083                 }
00084
00085                 return null;
00086             };
00087         }
00088     }
00089 }
```

8.23 ImageURIParser.cs

```
00001 /*
```

```
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020  Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Affero General Public License as
00007 published by the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012 GNU Affero General Public License for more details.
00013
00014 You should have received a copy of the GNU Affero General Public License
00015 along with this program.    If not, see <http://www.gnu.org/licenses/>
00016 */
00017
00018 using MuPDFCore;
00019 using System;
00020 using System.IO;
00021 using System.Linq;
00022 using System.Runtime.InteropServices;
00023
00024 namespace VectSharp.MuPDFUtils
00025 {
00026 /// <summary>
00027 /// Provides a method to parse an image URI into a page.
00028 /// </summary>
00029     public static class ImageURIParser
00030     {
00031 /// <summary>
00032 /// Parses an image URI into a page.  This is intended to replace the default image URI interpreter in
      <c>VectSharp.SVG.Parser.ParseImageURI</c>.  To do this, use something like:
00033 /// <code>VectSharp.SVG.Parser.ParseImageURI =
      VectSharp.MuPDFUtils.ImageURIParser.Parser(VectSharp.SVG.Parser.ParseSVGURI);</code>
00034 /// </summary>
00035 /// <param name="parseSVG">A function to parse an SVG image uri into a page.  You should pass
      <c>VectSharp.SVG.Parser.ParseSVGURI</c> as this argument.</param>
00036 /// <returns>A function to parse an image URI into a page.</returns>
00037         public static Func<string, bool, Page> Parser(Func<string, bool, Page> parseSVG)
00038         {
00039             return (string uri, bool interpolate) =>
00040             {
00041                 if (uri.StartsWith("data:"))
00042                 {
00043                     string mimeType = uri.Substring(uri.IndexOf(":") + 1, uri.IndexOf(";") -
      uri.IndexOf(":") - 1);
00044
00045                     string type = uri.Substring(uri.IndexOf(";") + 1, uri.IndexOf(",") -
      uri.IndexOf(";") - 1);
00046
00047                     if (mimeType != "image/svg+xml")
00048                     {
00049                         int offset = uri.IndexOf(",") + 1;
00050
00051                         byte[] parsed;
00052
00053                         bool isVector = false;
00054
00055                         InputFileTypes fileType;
00056
00057                         switch (mimeType)
00058                         {
00059                             case "image/png":
00060                                 fileType = InputFileTypes.PNG;
00061                                 break;
00062                             case "image/jpeg":
00063                             case "image/jpg":
00064                                 fileType = InputFileTypes.JPEG;
00065                                 break;
00066                             case "image/gif":
00067                                 fileType = InputFileTypes.GIF;
00068                                 break;
00069                             case "image/bmp":
00070                                 fileType = InputFileTypes.BMP;
00071                                 break;
00072                             case "image/tiff":
00073                             case "image/tif":
00074                                 fileType = InputFileTypes.TIFF;
00075                                 break;
00076                             case "application/oxps":
00077                             case "application/vnd.ms-xpsdocument":
00078                                 fileType = InputFileTypes.XPS;
00079                                 isVector = true;
00080                                 break;
00081                             case "application/x-cbz":
00082                                 fileType = InputFileTypes.CBZ;
00083                                 break;
```

```
00084                               case "application/epub+zip":
00085                                   fileType = InputFileTypes.EPUB;
00086                                   isVector = true;
00087                                   break;
00088                               case "text/fb2+xml":
00089                                   fileType = InputFileTypes.FB2;
00090                                   break;
00091                               case "image/x-portable-anymap":
00092                                   fileType = InputFileTypes.PNM;
00093                                   break;
00094                               case "image/x-portable-arbitrarymap":
00095                                   fileType = InputFileTypes.PAM;
00096                                   break;
00097                               case "application/pdf":
00098                                   fileType = InputFileTypes.PDF;
00099                                   isVector = true;
00100                                   break;
00101                               default:
00102                                   fileType = InputFileTypes.PDF;
00103                                   break;
00104                           }
00105
00106                           string substring = uri.Substring(offset);
00107
00108
00109                           switch (type)
00110                           {
00111                               case "base64":
00112                                   parsed = Convert.FromBase64String(uri.Substring(offset));
00113                                   break;
00114                               case "":
00115                                   parsed = (from el in
      System.Web.HttpUtility.UrlDecode(uri.Substring(offset)) select (byte)el).ToArray();
00116                                   break;
00117                               default:
00118                                   throw new InvalidDataException("Unknown data stream type!");
00119                           }
00120
00121                           if (!isVector)
00122                           {
00123                               GCHandle handle = GCHandle.Alloc(parsed, GCHandleType.Pinned);
00124
00125                               RasterImageStream img = new RasterImageStream(handle.AddrOfPinnedObject(),
      parsed.Length, fileType, interpolate:  interpolate);
00126
00127                               handle.Free();
00128
00129                               Page pag = new Page(img.Width, img.Height);
00130
00131                               pag.Graphics.DrawRasterImage(0, 0, img);
00132
00133                               return pag;
00134                           }
00135                           else
00136                           {
00137                               string tempFile = Path.GetTempFileName();
00138
00139                               using (MuPDFContext context = new MuPDFContext())
00140                               {
00141                                   using (MuPDFDocument document = new MuPDFDocument(context, parsed,
      fileType))
00142                                   {
00143                                       MuPDFDocument.CreateDocument(context, tempFile,
      DocumentOutputFileTypes.SVG, true, document.Pages[0]);
00144                                   }
00145                               }
00146
00147                               string tbr = "data:image/svg+xml;," +
      System.Web.HttpUtility.UrlEncode(File.ReadAllText(tempFile));
00148
00149                               File.Delete(tempFile);
00150
00151                               return parseSVG(tbr, interpolate);
00152                           }
00153                       }
00154                       else
00155                       {
00156                           return parseSVG(uri, interpolate);
00157                       }
00158                   }
00159
00160                   return null;
00161               };
00162           }
00163       }
00164   }
```

8.24 PDFParser.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2024  Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Affero General Public License as
00007 published by the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012 GNU Affero General Public License for more details.
00013
00014 You should have received a copy of the GNU Affero General Public License
00015 along with this program.    If not, see <http://www.gnu.org/licenses/>
00016 */
00017
00018 using MuPDFCore;
00019 using System;
00020 using System.Collections.Generic;
00021 using System.IO;
00022 using VectSharp.SVG;
00023
00024 namespace VectSharp.MuPDFUtils
00025 {
00026 /// <summary>
00027 /// Contains methods to import PDF documents as vectors.
00028 /// </summary>
00029     public static class PDFParser
00030     {
00031         private static Page FromMuPDFDocument(MuPDFContext context, MuPDFDocument document, int
      pageNumber, SVGCreationOptions.TextOption textOption, bool includeAnnotations, Dictionary<string,
      MuPDFLinkDestination> linkDestinations)
00032         {
00033             string tempFile = Path.GetTempFileName();
00034
00035             try
00036             {
00037                 MuPDFDocument.Create.SVGDocument(context, tempFile, document.Pages[pageNumber], new
      SVGCreationOptions() { TextRendering = textOption, IncludeAnnotations = includeAnnotations });
00038                 Parser.ParseImageURI = ImageURIParser.Parser(Parser.ParseSVGURI);
00039
00040                 Page parsedPage = Parser.FromFile(tempFile);
00041
00042                 if (linkDestinations != null)
00043                 {
00044                     foreach (MuPDFLink link in document.Pages[pageNumber].Links)
00045                     {
00046                         if (link.IsVisible)
00047                         {
00048                             string tag = Guid.NewGuid().ToString("N");
00049                             parsedPage.Graphics.FillRectangle(link.ActiveArea.X0, link.ActiveArea.Y0,
      link.ActiveArea.Width, link.ActiveArea.Height, Colour.FromRgba(0, 0, 0, 0), tag:  tag);
00050                             linkDestinations.Add(tag, link.Destination);
00051                         }
00052                     }
00053                 }
00054
00055                 return parsedPage;
00056             }
00057             finally
00058             {
00059                 File.Delete(tempFile);
00060             }
00061         }
00062
00063         private static Dictionary<string, string> SetUpLinks(Page page, int pageNumber,
      Dictionary<string, MuPDFLinkDestination> linkDestinations)
00064         {
00065             Dictionary<string, string> tbr = new Dictionary<string, string>();
00066
00067             foreach (KeyValuePair<string, MuPDFLinkDestination> link in linkDestinations)
00068             {
00069                 if (link.Value.Type == MuPDFLinkDestination.DestinationType.External)
00070                 {
00071                     tbr.Add(link.Key, (link.Value as MuPDFExternalLinkDestination).Uri);
00072                 }
00073                 else if (link.Value.Type == MuPDFLinkDestination.DestinationType.Internal)
00074                 {
00075                     MuPDFInternalLinkDestination intern = link.Value as MuPDFInternalLinkDestination;
00076                     if (intern.PageNumber == pageNumber)
00077                     {
00078                         string tag = Guid.NewGuid().ToString("N");
00079
00080                         switch (intern.InternalType)
```

```
00081                         {
00082                             case MuPDFInternalLinkDestination.InternalDestinationType.XYZoom:
00083                                 page.Graphics.FillRectangle(intern.X, intern.Y, 1, 1,
      Colour.FromRgba(0, 0, 0, 0), tag);
00084                                 break;
00085                             case MuPDFInternalLinkDestination.InternalDestinationType.FitRectangle:
00086                                 page.Graphics.FillRectangle(intern.X, intern.Y, intern.Width,
      intern.Height, Colour.FromRgba(0, 0, 0, 0), tag);
00087                                 break;
00088                             case MuPDFInternalLinkDestination.InternalDestinationType.Fit:
00089                             case MuPDFInternalLinkDestination.InternalDestinationType.FitBoundingBox:
00090                                 page.Graphics.FillRectangle(0, 0, page.Width, page.Height,
      Colour.FromRgba(0, 0, 0, 0), tag);
00091                                 break;
00092                             case MuPDFInternalLinkDestination.InternalDestinationType.FitWidth:
00093                             case
      MuPDFInternalLinkDestination.InternalDestinationType.FitBoundingBoxWidth:
00094                                 page.Graphics.FillRectangle(0, intern.Y, page.Width, page.Height -
      intern.Y, Colour.FromRgba(0, 0, 0, 0), tag);
00095                                 break;
00096                             case MuPDFInternalLinkDestination.InternalDestinationType.FitHeight:
00097                             case
      MuPDFInternalLinkDestination.InternalDestinationType.FitBoundingBoxHeight:
00098                                 page.Graphics.FillRectangle(intern.X, 0, page.Width - intern.X,
      page.Height, Colour.FromRgba(0, 0, 0, 0), tag);
00099                                 break;
00100                         }
00101
00102                         tbr.Add(link.Key, "#" + tag);
00103                     }
00104                 }
00105             }
00106
00107             return tbr;
00108         }
00109
00110         private static Dictionary<string, string> SetUpLinks(IReadOnlyList<Page> pages,
      Dictionary<string, MuPDFLinkDestination> linkDestinations)
00111         {
00112             Dictionary<string, string> tbr = new Dictionary<string, string>();
00113
00114             foreach (KeyValuePair<string, MuPDFLinkDestination> link in linkDestinations)
00115             {
00116                 if (link.Value.Type == MuPDFLinkDestination.DestinationType.External)
00117                 {
00118                     tbr.Add(link.Key, (link.Value as MuPDFExternalLinkDestination).Uri);
00119                 }
00120                 else if (link.Value.Type == MuPDFLinkDestination.DestinationType.Internal)
00121                 {
00122                     MuPDFInternalLinkDestination intern = link.Value as MuPDFInternalLinkDestination;
00123                     Page page = pages[intern.PageNumber];
00124
00125                     string tag = Guid.NewGuid().ToString("N");
00126
00127                     switch (intern.InternalType)
00128                     {
00129                         case MuPDFInternalLinkDestination.InternalDestinationType.XYZoom:
00130                             page.Graphics.FillRectangle(intern.X, intern.Y, 1, 1, Colour.FromRgba(0,
      0, 0, 0), tag);
00131                             break;
00132                         case MuPDFInternalLinkDestination.InternalDestinationType.FitRectangle:
00133                             page.Graphics.FillRectangle(intern.X, intern.Y, intern.Width,
      intern.Height, Colour.FromRgba(0, 0, 0, 0), tag);
00134                             break;
00135                         case MuPDFInternalLinkDestination.InternalDestinationType.Fit:
00136                         case MuPDFInternalLinkDestination.InternalDestinationType.FitBoundingBox:
00137                             page.Graphics.FillRectangle(0, 0, page.Width, page.Height,
      Colour.FromRgba(0, 0, 0, 0), tag);
00138                             break;
00139                         case MuPDFInternalLinkDestination.InternalDestinationType.FitWidth:
00140                         case MuPDFInternalLinkDestination.InternalDestinationType.FitBoundingBoxWidth:
00141                             page.Graphics.FillRectangle(0, intern.Y, page.Width, page.Height -
      intern.Y, Colour.FromRgba(0, 0, 0, 0), tag);
00142                             break;
00143                         case MuPDFInternalLinkDestination.InternalDestinationType.FitHeight:
00144                         case
      MuPDFInternalLinkDestination.InternalDestinationType.FitBoundingBoxHeight:
00145                             page.Graphics.FillRectangle(intern.X, 0, page.Width - intern.X,
      page.Height, Colour.FromRgba(0, 0, 0, 0), tag);
00146                             break;
00147                     }
00148
00149                     tbr.Add(link.Key, "#" + tag);
00150                 }
00151             }
00152
00153             return tbr;
```

```
00154         }
00155
00156
00157 /// <summary>
00158 /// Import the specified page from a document.
00159 /// </summary>
00160 /// <param name="fileName">The path to the document file.</param>
00161 /// <param name="pageNumber">The page number (starting at 0).</param>
00162 /// <param name="textOption">This parameter determines whether text elements are rendered as paths
      (which provides more accurate results) or as text.</param>
00163 /// <param name="includeAnnotations">Whether annotations (e.g., signatures) should be included in the
      imported document.</param>
00164 /// <returns>A <see cref="Page"/> containing a representation of the specified page from the
      document.</returns>
00165         public static Page FromFile(string fileName, int pageNumber, SVGCreationOptions.TextOption
      textOption = SVGCreationOptions.TextOption.TextAsPath, bool includeAnnotations = true)
00166         {
00167             using (MuPDFContext context = new MuPDFContext())
00168             {
00169                 using (MuPDFDocument document = new MuPDFDocument(context, fileName))
00170                 {
00171                     return FromMuPDFDocument(context, document, pageNumber, textOption,
      includeAnnotations, null);
00172                 }
00173             }
00174         }
00175
00176 /// <summary>
00177 /// Import the specified page from a document.
00178 /// </summary>
00179 /// <param name="fileName">The path to the document file.</param>
00180 /// <param name="pageNumber">The page number (starting at 0).</param>
00181 /// <param name="linkDestinations">When this method returns, this variable will contain a dictionary
      used to associate graphic action tags to hyperlinks.  This can be used to enable such links when
      rendering the <see cref="Page"/> to a file.</param>
00182 /// <param name="textOption">This parameter determines whether text elements are rendered as paths
      (which provides more accurate results) or as text.</param>
00183 /// <param name="includeAnnotations">Whether annotations (e.g., signatures) should be included in the
      imported document.</param>
00184 /// <returns>A <see cref="Page"/> containing a representation of the specified page from the
      document.</returns>
00185         public static Page FromFile(string fileName, int pageNumber, out Dictionary<string, string>
      linkDestinations, SVGCreationOptions.TextOption textOption = SVGCreationOptions.TextOption.TextAsPath,
      bool includeAnnotations = true)
00186         {
00187             using (MuPDFContext context = new MuPDFContext())
00188             {
00189                 using (MuPDFDocument document = new MuPDFDocument(context, fileName))
00190                 {
00191                     Dictionary<string, MuPDFLinkDestination> dests = new Dictionary<string,
      MuPDFLinkDestination>();
00192                     Page pag = FromMuPDFDocument(context, document, pageNumber, textOption,
      includeAnnotations, dests);
00193                     linkDestinations = SetUpLinks(pag, pageNumber, dests);
00194                     return pag;
00195                 }
00196             }
00197         }
00198
00199 /// <summary>
00200 /// Import the specified page from a document.
00201 /// </summary>
00202 /// <param name="stream">The <see cref="Stream"/> containing the document to import.</param>
00203 /// <param name="pageNumber">The page number (starting at 0).</param>
00204 /// <param name="fileType">The type of document contained in the <see cref="Stream"/>.</param>
00205 /// <param name="textOption">This parameter determines whether text elements are rendered as paths
      (which provides more accurate results) or as text.</param>
00206 /// <param name="includeAnnotations">Whether annotations (e.g., signatures) should be included in the
      imported document.</param>
00207 /// <returns>A <see cref="Page"/> containing a representation of the specified page from the
      document.</returns>
00208         public static Page FromStream(Stream stream, int pageNumber, InputFileTypes fileType =
      InputFileTypes.PDF, SVGCreationOptions.TextOption textOption =
      SVGCreationOptions.TextOption.TextAsPath, bool includeAnnotations = true)
00209         {
00210             MemoryStream ms = new MemoryStream();
00211             stream.CopyTo(ms);
00212
00213             using (MuPDFContext context = new MuPDFContext())
00214             {
00215                 using (MuPDFDocument document = new MuPDFDocument(context, ref ms, fileType))
00216                 {
00217                     return FromMuPDFDocument(context, document, pageNumber, textOption,
      includeAnnotations, null);
00218                 }
00219             }
00220         }
```

```
00221
00222 /// <summary>
00223 /// Import the specified page from a document.
00224 /// </summary>
00225 /// <param name="stream">The <see cref="Stream"/> containing the document to import.</param>
00226 /// <param name="pageNumber">The page number (starting at 0).</param>
00227 /// <param name="linkDestinations">When this method returns, this variable will contain a dictionary
      used to associate graphic action tags to hyperlinks.  This can be used to enable such links when
      rendering the <see cref="Page"/> to a file.</param>
00228 /// <param name="fileType">The type of document contained in the <see cref="Stream"/>.</param>
00229 /// <param name="textOption">This parameter determines whether text elements are rendered as paths
      (which provides more accurate results) or as text.</param>
00230 /// <param name="includeAnnotations">Whether annotations (e.g., signatures) should be included in the
      imported document.</param>
00231 /// <returns>A <see cref="Page"/> containing a representation of the specified page from the
      document.</returns>
00232         public static Page FromStream(Stream stream, int pageNumber, out Dictionary<string, string>
      linkDestinations, InputFileTypes fileType = InputFileTypes.PDF, SVGCreationOptions.TextOption
      textOption = SVGCreationOptions.TextOption.TextAsPath, bool includeAnnotations = true)
00233         {
00234             MemoryStream ms = new MemoryStream();
00235             stream.CopyTo(ms);
00236
00237             using (MuPDFContext context = new MuPDFContext())
00238             {
00239                 using (MuPDFDocument document = new MuPDFDocument(context, ref ms, fileType))
00240                 {
00241                     Dictionary<string, MuPDFLinkDestination> dests = new Dictionary<string,
      MuPDFLinkDestination>();
00242                     Page pag = FromMuPDFDocument(context, document, pageNumber, textOption,
      includeAnnotations, dests);
00243                     linkDestinations = SetUpLinks(pag, pageNumber, dests);
00244                     return pag;
00245                 }
00246             }
00247         }
00248
00249 /// <summary>
00250 /// Import all pages from a document.
00251 /// </summary>
00252 /// <param name="fileName">The path to the document file.</param>
00253 /// <param name="textOption">This parameter determines whether text elements are rendered as paths
      (which provides more accurate results) or as text.</param>
00254 /// <param name="includeAnnotations">Whether annotations (e.g., signatures) should be included in the
      imported document.</param>
00255 /// <returns>A <see cref="Document"/> containing a representation of every page from the
      document.</returns>
00256         public static Document FromFile(string fileName, SVGCreationOptions.TextOption textOption =
      SVGCreationOptions.TextOption.TextAsPath, bool includeAnnotations = true)
00257         {
00258             using (MuPDFContext context = new MuPDFContext())
00259             {
00260                 using (MuPDFDocument document = new MuPDFDocument(context, fileName))
00261                 {
00262                     Document doc = new Document();
00263
00264                     for (int i = 0; i < document.Pages.Count; i++)
00265                     {
00266                         doc.Pages.Add(FromMuPDFDocument(context, document, i, textOption,
      includeAnnotations, null));
00267                     }
00268
00269                     return doc;
00270                 }
00271             }
00272         }
00273
00274 /// <summary>
00275 /// Import all pages from a document.
00276 /// </summary>
00277 /// <param name="fileName">The path to the document file.</param>
00278 /// <param name="linkDestinations">When this method returns, this variable will contain a dictionary
      used to associate graphic action tags to hyperlinks.  This can be used to enable such links when
      rendering the <see cref="Page"/> to a file.</param>
00279 /// <param name="textOption">This parameter determines whether text elements are rendered as paths
      (which provides more accurate results) or as text.</param>
00280 /// <param name="includeAnnotations">Whether annotations (e.g., signatures) should be included in the
      imported document.</param>
00281 /// <returns>A <see cref="Document"/> containing a representation of every page from the
      document.</returns>
00282         public static Document FromFile(string fileName, out Dictionary<string, string>
      linkDestinations, SVGCreationOptions.TextOption textOption = SVGCreationOptions.TextOption.TextAsPath,
      bool includeAnnotations = true)
00283         {
00284             using (MuPDFContext context = new MuPDFContext())
00285             {
00286                 using (MuPDFDocument document = new MuPDFDocument(context, fileName))
```

```
00287                 {
00288                     Document doc = new Document();
00289 
00290                     Dictionary<string, MuPDFLinkDestination> dest = new Dictionary<string,
      MuPDFLinkDestination>();
00291 
00292                     for (int i = 0; i < document.Pages.Count; i++)
00293                     {
00294                         doc.Pages.Add(FromMuPDFDocument(context, document, i, textOption,
      includeAnnotations, dest));
00295                     }
00296 
00297                     linkDestinations = SetUpLinks(doc.Pages, dest);
00298 
00299                     return doc;
00300                 }
00301             }
00302         }
00303 
00304 /// <summary>
00305 /// Import all pages from a document.
00306 /// </summary>
00307 /// <param name="stream">The <see cref="Stream"/> containing the document to import.</param>
00308 /// <param name="fileType">The type of document contained in the <see cref="Stream"/>.</param>
00309 /// <param name="textOption">This parameter determines whether text elements are rendered as paths
      (which provides more accurate results) or as text.</param>
00310 /// <param name="includeAnnotations">Whether annotations (e.g., signatures) should be included in the
      imported document.</param>
00311 /// <returns>A <see cref="Document"/> containing a representation of every page from the
      document.</returns>
00312         public static Document FromStream(Stream stream, InputFileTypes fileType = InputFileTypes.PDF,
      SVGCreationOptions.TextOption textOption = SVGCreationOptions.TextOption.TextAsPath, bool
      includeAnnotations = true)
00313         {
00314             MemoryStream ms = new MemoryStream();
00315             stream.CopyTo(ms);
00316 
00317             using (MuPDFContext context = new MuPDFContext())
00318             {
00319                 using (MuPDFDocument document = new MuPDFDocument(context, ref ms, fileType))
00320                 {
00321                     Document doc = new Document();
00322 
00323                     for (int i = 0; i < document.Pages.Count; i++)
00324                     {
00325                         doc.Pages.Add(FromMuPDFDocument(context, document, i, textOption,
      includeAnnotations, null));
00326                     }
00327 
00328                     return doc;
00329                 }
00330             }
00331         }
00332 
00333 /// <summary>
00334 /// Import all pages from a document.
00335 /// </summary>
00336 /// <param name="stream">The <see cref="Stream"/> containing the document to import.</param>
00337 /// <param name="linkDestinations">When this method returns, this variable will contain a dictionary
      used to associate graphic action tags to hyperlinks.  This can be used to enable such links when
      rendering the <see cref="Page"/> to a file.</param>
00338 /// <param name="fileType">The type of document contained in the <see cref="Stream"/>.</param>
00339 /// <param name="textOption">This parameter determines whether text elements are rendered as paths
      (which provides more accurate results) or as text.</param>
00340 /// <param name="includeAnnotations">Whether annotations (e.g., signatures) should be included in the
      imported document.</param>
00341 /// <returns>A <see cref="Document"/> containing a representation of every page from the
      document.</returns>
00342         public static Document FromStream(Stream stream, out Dictionary<string, string>
      linkDestinations, InputFileTypes fileType = InputFileTypes.PDF, SVGCreationOptions.TextOption
      textOption = SVGCreationOptions.TextOption.TextAsPath, bool includeAnnotations = true)
00343         {
00344             MemoryStream ms = new MemoryStream();
00345             stream.CopyTo(ms);
00346 
00347             using (MuPDFContext context = new MuPDFContext())
00348             {
00349                 using (MuPDFDocument document = new MuPDFDocument(context, ref ms, fileType))
00350                 {
00351                     Document doc = new Document();
00352 
00353                     Dictionary<string, MuPDFLinkDestination> dest = new Dictionary<string,
      MuPDFLinkDestination>();
00354 
00355                     for (int i = 0; i < document.Pages.Count; i++)
00356                     {
00357                         doc.Pages.Add(FromMuPDFDocument(context, document, i, textOption,
```

```
    includeAnnotations, dest));
00358                     }
00359
00360                     linkDestinations = SetUpLinks(doc.Pages, dest);
00361
00362                     return doc;
00363                 }
00364             }
00365         }
00366     }
00367 }
```

8.25 RasterImages.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using SixLabors.ImageSharp;
00019 using SixLabors.ImageSharp.Advanced;
00020 using SixLabors.ImageSharp.Processing;
00021 using System;
00022 using System.IO;
00023 using System.Runtime.InteropServices;
00024
00025 namespace VectSharp.ImageSharpUtils
00026 {
00027 /// <summary>
00028 /// A <see cref="RasterImage"/> created from a file.
00029 /// </summary>
00030     public class RasterImageFile :  RasterImage
00031     {
00032 /// <summary>
00033 /// Creates a new <see cref="RasterImage"/> from the specified file.
00034 /// </summary>
00035 /// <param name="fileName">The path to the file containing the image.</param>
00036 /// <param name="alpha">A boolean value indicating whether transparency (alpha) data from the image
      should be preserved or not.</param>
00037 /// <param name="interpolate">A boolean value indicating whether the image should be interpolated when
      it is resized or not.</param>
00038         public RasterImageFile(string fileName, bool alpha = true, bool interpolate = true)
00039         {
00040             Image image;
00041
00042             if (alpha)
00043             {
00044                 image = Image.Load<SixLabors.ImageSharp.PixelFormats.Rgba32>(fileName);
00045             }
00046             else
00047             {
00048                 image = Image.Load<SixLabors.ImageSharp.PixelFormats.Rgb24>(fileName);
00049             }
00050
00051             image.Mutate(x => x.AutoOrient());
00052
00053             int stride = image.Width * (alpha ?  4 :  3);
00054             int size = stride * image.Height;
00055
00056             IntPtr tbr = Marshal.AllocHGlobal(size);
00057             GC.AddMemoryPressure(size);
00058
00059             IntPtr pointer = tbr;
00060
00061             unsafe
00062             {
00063                 if (alpha)
00064                 {
00065                     Image<SixLabors.ImageSharp.PixelFormats.Rgba32> img =
      (Image<SixLabors.ImageSharp.PixelFormats.Rgba32>)image;
00066
```

```
00067                     for (int y = 0; y < image.Height; y++)
00068                     {
00069                         Memory<SixLabors.ImageSharp.PixelFormats.Rgba32> row =
      img.DangerousGetPixelRowMemory(y);
00070 
00071                         Span<SixLabors.ImageSharp.PixelFormats.Rgba32> newRow = new
      Span<SixLabors.ImageSharp.PixelFormats.Rgba32>(pointer.ToPointer(), row.Length);
00072                         row.Span.CopyTo(newRow);
00073 
00074                         pointer = IntPtr.Add(pointer, stride);
00075                     }
00076                 }
00077                 else
00078                 {
00079                     Image<SixLabors.ImageSharp.PixelFormats.Rgb24> img =
      (Image<SixLabors.ImageSharp.PixelFormats.Rgb24>)image;
00080 
00081                     for (int y = 0; y < image.Height; y++)
00082                     {
00083                         Memory<SixLabors.ImageSharp.PixelFormats.Rgb24> row =
      img.DangerousGetPixelRowMemory(y);
00084 
00085                         Span<SixLabors.ImageSharp.PixelFormats.Rgb24> newRow = new
      Span<SixLabors.ImageSharp.PixelFormats.Rgb24>(pointer.ToPointer(), row.Length);
00086                         row.Span.CopyTo(newRow);
00087 
00088                         pointer = IntPtr.Add(pointer, stride);
00089                     }
00090                 }
00091             }
00092 
00093             this.Width = image.Width;
00094             this.Height = image.Height;
00095             this.HasAlpha = alpha;
00096             this.Interpolate = interpolate;
00097             this.Id = Guid.NewGuid().ToString();
00098             this.ImageDataAddress = tbr;
00099             this.DataHolder = new DisposableIntPtr(tbr);
00100         }
00101     }
00102 
00103 /// <summary>
00104 /// A <see cref="RasterImage"/> created from a stream.
00105 /// </summary>
00106     public class RasterImageStream :  RasterImage
00107     {
00108 /// <summary>
00109 /// Creates a new <see cref="RasterImage"/> from the specified stream.
00110 /// </summary>
00111 /// <param name="imageStream">The stream containing the image data.</param>
00112 /// <param name="alpha">A boolean value indicating whether transparency (alpha) data from the image
      should be preserved or not.</param>
00113 /// <param name="interpolate">A boolean value indicating whether the image should be interpolated when
      it is resized or not.</param>
00114         public RasterImageStream(Stream imageStream, bool alpha = true, bool interpolate = true)
00115         {
00116             Image image;
00117 
00118             if (alpha)
00119             {
00120                 image = Image.Load<SixLabors.ImageSharp.PixelFormats.Rgba32>(imageStream);
00121             }
00122             else
00123             {
00124                 image = Image.Load<SixLabors.ImageSharp.PixelFormats.Rgb24>(imageStream);
00125             }
00126 
00127             image.Mutate(x => x.AutoOrient());
00128 
00129             int stride = image.Width * (alpha ?  4 :  3);
00130             int size = stride * image.Height;
00131 
00132             IntPtr tbr = Marshal.AllocHGlobal(size);
00133             GC.AddMemoryPressure(size);
00134 
00135             IntPtr pointer = tbr;
00136 
00137             unsafe
00138             {
00139                 if (alpha)
00140                 {
00141                     Image<SixLabors.ImageSharp.PixelFormats.Rgba32> img =
      (Image<SixLabors.ImageSharp.PixelFormats.Rgba32>)image;
00142 
00143                     for (int y = 0; y < image.Height; y++)
00144                     {
00145                         Memory<SixLabors.ImageSharp.PixelFormats.Rgba32> row =
```

```
img.DangerousGetPixelRowMemory(y);
00146
00147                         Span<SixLabors.ImageSharp.PixelFormats.Rgba32> newRow = new
Span<SixLabors.ImageSharp.PixelFormats.Rgba32>(pointer.ToPointer(), row.Length);
00148                         row.Span.CopyTo(newRow);
00149
00150                         pointer = IntPtr.Add(pointer, stride);
00151                     }
00152                 }
00153                 else
00154                 {
00155                     Image<SixLabors.ImageSharp.PixelFormats.Rgb24> img =
(Image<SixLabors.ImageSharp.PixelFormats.Rgb24>)image;
00156
00157                     for (int y = 0; y < image.Height; y++)
00158                     {
00159                         Memory<SixLabors.ImageSharp.PixelFormats.Rgb24> row =
img.DangerousGetPixelRowMemory(y);
00160
00161                         Span<SixLabors.ImageSharp.PixelFormats.Rgb24> newRow = new
Span<SixLabors.ImageSharp.PixelFormats.Rgb24>(pointer.ToPointer(), row.Length);
00162                         row.Span.CopyTo(newRow);
00163
00164                         pointer = IntPtr.Add(pointer, stride);
00165                     }
00166                 }
00167             }
00168
00169             this.Width = image.Width;
00170             this.Height = image.Height;
00171             this.HasAlpha = alpha;
00172             this.Interpolate = interpolate;
00173             this.Id = Guid.NewGuid().ToString();
00174             this.ImageDataAddress = tbr;
00175             this.DataHolder = new DisposableIntPtr(tbr);
00176         }
00177
00178 /// <summary>
00179 /// Creates a new <see cref="RasterImage"/> from the specified stream.
00180 /// </summary>
00181 /// <param name="imageAddress">A pointer to the address where the image data is contained.</param>
00182 /// <param name="imageLength">The length in bytes of the image data.</param>
00183 /// <param name="alpha">A boolean value indicating whether transparency (alpha) data from the image
should be preserved or not.</param>
00184 /// <param name="interpolate">A boolean value indicating whether the image should be interpolated when
it is resized or not.</param>
00185         public RasterImageStream(IntPtr imageAddress, int imageLength, bool alpha = true, bool
interpolate = true)
00186         {
00187             unsafe
00188             {
00189                 ReadOnlySpan<byte> imageSpan = new ReadOnlySpan<byte>((void*)imageAddress,
imageLength);
00190
00191                 Image image;
00192
00193                 if (alpha)
00194                 {
00195                     image = Image.Load<SixLabors.ImageSharp.PixelFormats.Rgba32>(imageSpan);
00196                 }
00197                 else
00198                 {
00199                     image = Image.Load<SixLabors.ImageSharp.PixelFormats.Rgb24>(imageSpan);
00200                 }
00201
00202                 image.Mutate(x => x.AutoOrient());
00203
00204                 int stride = image.Width * (alpha ?  4 :  3);
00205                 int size = stride * image.Height;
00206
00207                 IntPtr tbr = Marshal.AllocHGlobal(size);
00208                 GC.AddMemoryPressure(size);
00209
00210                 IntPtr pointer = tbr;
00211
00212                 if (alpha)
00213                 {
00214                     Image<SixLabors.ImageSharp.PixelFormats.Rgba32> img =
(Image<SixLabors.ImageSharp.PixelFormats.Rgba32>)image;
00215
00216                     for (int y = 0; y < image.Height; y++)
00217                     {
00218                         Memory<SixLabors.ImageSharp.PixelFormats.Rgba32> row =
img.DangerousGetPixelRowMemory(y);
00219
00220                         Span<SixLabors.ImageSharp.PixelFormats.Rgba32> newRow = new
Span<SixLabors.ImageSharp.PixelFormats.Rgba32>(pointer.ToPointer(), row.Length);
```

```
00221                         row.Span.CopyTo(newRow);
00222
00223                         pointer = IntPtr.Add(pointer, stride);
00224                     }
00225                 }
00226                 else
00227                 {
00228                     Image<SixLabors.ImageSharp.PixelFormats.Rgb24> img =
    (Image<SixLabors.ImageSharp.PixelFormats.Rgb24>)image;
00229
00230                     for (int y = 0; y < image.Height; y++)
00231                     {
00232                         Memory<SixLabors.ImageSharp.PixelFormats.Rgb24> row =
    img.DangerousGetPixelRowMemory(y);
00233
00234                         Span<SixLabors.ImageSharp.PixelFormats.Rgb24> newRow = new
    Span<SixLabors.ImageSharp.PixelFormats.Rgb24>(pointer.ToPointer(), row.Length);
00235                         row.Span.CopyTo(newRow);
00236
00237                         pointer = IntPtr.Add(pointer, stride);
00238                     }
00239                 }
00240
00241                 this.Width = image.Width;
00242                 this.Height = image.Height;
00243                 this.HasAlpha = alpha;
00244                 this.Interpolate = interpolate;
00245                 this.Id = Guid.NewGuid().ToString();
00246                 this.ImageDataAddress = tbr;
00247                 this.DataHolder = new DisposableIntPtr(tbr);
00248             }
00249         }
00250     }
00251 }
```

8.26 RasterImages.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020  Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Affero General Public License as
00007 published by the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012 GNU Affero General Public License for more details.
00013
00014 You should have received a copy of the GNU Affero General Public License
00015 along with this program.    If not, see <http://www.gnu.org/licenses/>
00016 */
00017
00018 using MuPDFCore;
00019 using System;
00020 using System.IO;
00021 using System.Runtime.InteropServices;
00022
00023 namespace VectSharp.MuPDFUtils
00024 {
00025 /// <summary>
00026 /// A <see cref="RasterImage"/> created from a file.
00027 /// </summary>
00028     public class RasterImageFile :  RasterImage
00029     {
00030 /// <summary>
00031 /// Creates a new <see cref="RasterImage"/> from the specified file.
00032 /// </summary>
00033 /// <param name="fileName">The path to the file containing the image.</param>
00034 /// <param name="pageNumber">The number of the page in the file from which the image should be
    created, starting at 0.  Only useful for multi-page formats, such as PDF.</param>
00035 /// <param name="scale">The scale factor at which to render the image.</param>
00036 /// <param name="alpha">A boolean value indicating whether transparency (alpha) data from the image
    should be preserved or not.</param>
00037 /// <param name="interpolate">A boolean value indicating whether the image should be interpolated when
    it is resized or not.</param>
00038         public RasterImageFile(string fileName, int pageNumber = 0, double scale = 1, bool alpha =
    true, bool interpolate = true)
00039         {
00040             using (MuPDFContext context = new MuPDFContext())
00041             {
00042                 using (MuPDFDocument document = new MuPDFDocument(context, fileName))
```

```
00043                 {
00044                     RoundedRectangle roundedBounds = document.Pages[pageNumber].Bounds.Round(scale);
00045 
00046                     this.Width = roundedBounds.Width;
00047                     this.Height = roundedBounds.Height;
00048 
00049                     this.HasAlpha = alpha;
00050                     this.Interpolate = interpolate;
00051 
00052                     this.Id = Guid.NewGuid().ToString();
00053 
00054                     int imageSize = document.GetRenderedSize(pageNumber, scale, alpha ?
      MuPDFCore.PixelFormats.RGBA : MuPDFCore.PixelFormats.RGB);
00055 
00056                     this.ImageDataAddress = Marshal.AllocHGlobal(imageSize);
00057                     this.DataHolder = new DisposableIntPtr(this.ImageDataAddress);
00058                     GC.AddMemoryPressure(imageSize);
00059 
00060                     document.Render(pageNumber, scale, alpha ?  MuPDFCore.PixelFormats.RGBA :
      MuPDFCore.PixelFormats.RGB, this.ImageDataAddress);
00061                 }
00062             }
00063         }
00064     }
00065 
00066 /// <summary>
00067 /// A <see cref="RasterImage"/> created from a stream.
00068 /// </summary>
00069     public class RasterImageStream :  RasterImage
00070     {
00071 /// <summary>
00072 /// Creates a new <see cref="RasterImage"/> from the specified stream.
00073 /// </summary>
00074 /// <param name="imageStream">The stream containing the image data.</param>
00075 /// <param name="fileType">The type of the image contained in the stream.</param>
00076 /// <param name="pageNumber">The number of the page in the file from which the image should be
      created, starting at 0.  Only useful for multi-page formats, such as PDF.</param>
00077 /// <param name="scale">The scale factor at which to render the image.</param>
00078 /// <param name="alpha">A boolean value indicating whether transparency (alpha) data from the image
      should be preserved or not.</param>
00079 /// <param name="interpolate">A boolean value indicating whether the image should be interpolated when
      it is resized or not.</param>
00080         public RasterImageStream(Stream imageStream, InputFileTypes fileType, int pageNumber = 0,
      double scale = 1, bool alpha = true, bool interpolate = true)
00081         {
00082             using (MuPDFContext context = new MuPDFContext())
00083             {
00084                 IntPtr originalImageAddress;
00085                 long originalImageLength;
00086 
00087                 IDisposable toBeDisposed = null;
00088                 GCHandle handleToFree;
00089 
00090                 if (imageStream is MemoryStream ms)
00091                 {
00092                     int origin = (int)ms.Seek(0, SeekOrigin.Begin);
00093                     originalImageLength = ms.Length;
00094 
00095                     handleToFree = GCHandle.Alloc(ms.GetBuffer(), GCHandleType.Pinned);
00096                     originalImageAddress = handleToFree.AddrOfPinnedObject();
00097                 }
00098                 else
00099                 {
00100                     MemoryStream mem = new MemoryStream((int)imageStream.Length);
00101                     imageStream.CopyTo(mem);
00102 
00103                     toBeDisposed = mem;
00104 
00105                     int origin = (int)mem.Seek(0, SeekOrigin.Begin);
00106                     originalImageLength = mem.Length;
00107 
00108                     handleToFree = GCHandle.Alloc(mem.GetBuffer(), GCHandleType.Pinned);
00109                     originalImageAddress = handleToFree.AddrOfPinnedObject();
00110                 }
00111 
00112                 using (MuPDFDocument document = new MuPDFDocument(context, originalImageAddress,
      originalImageLength, fileType))
00113                 {
00114                     RoundedRectangle roundedBounds = document.Pages[pageNumber].Bounds.Round(scale);
00115 
00116                     this.Width = roundedBounds.Width;
00117                     this.Height = roundedBounds.Height;
00118 
00119                     this.HasAlpha = alpha;
00120                     this.Interpolate = interpolate;
00121 
00122                     this.Id = Guid.NewGuid().ToString();
```

```
00123
00124                     int imageSize = document.GetRenderedSize(pageNumber, scale, alpha ?
      MuPDFCore.PixelFormats.RGBA : MuPDFCore.PixelFormats.RGB);
00125
00126                     this.ImageDataAddress = Marshal.AllocHGlobal(imageSize);
00127                     this.DataHolder = new DisposableIntPtr(this.ImageDataAddress);
00128                     GC.AddMemoryPressure(imageSize);
00129
00130                     document.Render(pageNumber, scale, alpha ?  MuPDFCore.PixelFormats.RGBA :
      MuPDFCore.PixelFormats.RGB, this.ImageDataAddress);
00131                 }
00132
00133                 handleToFree.Free();
00134                 toBeDisposed?.Dispose();
00135             }
00136         }
00137
00138 /// <summary>
00139 /// Creates a new <see cref="RasterImage"/> from the specified stream.
00140 /// </summary>
00141 /// <param name="imageAddress">A pointer to the address where the image data is contained.</param>
00142 /// <param name="imageLength">The length in bytes of the image data.</param>
00143 /// <param name="fileType">The type of the image contained in the stream.</param>
00144 /// <param name="pageNumber">The number of the page in the file from which the image should be
      created, starting at 0.  Only useful for multi-page formats, such as PDF.</param>
00145 /// <param name="scale">The scale factor at which to render the image.</param>
00146 /// <param name="alpha">A boolean value indicating whether transparency (alpha) data from the image
      should be preserved or not.</param>
00147 /// <param name="interpolate">A boolean value indicating whether the image should be interpolated when
      it is resized or not.</param>
00148         public RasterImageStream(IntPtr imageAddress, long imageLength, InputFileTypes fileType, int
      pageNumber = 0, double scale = 1, bool alpha = true, bool interpolate = true)
00149         {
00150             using (MuPDFContext context = new MuPDFContext())
00151             {
00152                 using (MuPDFDocument document = new MuPDFDocument(context, imageAddress, imageLength,
      fileType))
00153                 {
00154                     RoundedRectangle roundedBounds = document.Pages[pageNumber].Bounds.Round(scale);
00155
00156                     this.Width = roundedBounds.Width;
00157                     this.Height = roundedBounds.Height;
00158
00159                     this.HasAlpha = alpha;
00160                     this.Interpolate = interpolate;
00161
00162                     this.Id = Guid.NewGuid().ToString();
00163
00164                     int imageSize = document.GetRenderedSize(pageNumber, scale, alpha ?
      MuPDFCore.PixelFormats.RGBA : MuPDFCore.PixelFormats.RGB);
00165
00166                     this.ImageDataAddress = Marshal.AllocHGlobal(imageSize);
00167                     this.DataHolder = new DisposableIntPtr(this.ImageDataAddress);
00168                     GC.AddMemoryPressure(imageSize);
00169
00170                     document.Render(pageNumber, scale, alpha ?  MuPDFCore.PixelFormats.RGBA :
      MuPDFCore.PixelFormats.RGB, this.ImageDataAddress);
00171                 }
00172             }
00173         }
00174     }
00175 }
```

8.27 AnnotationStyles.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2024 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System.Collections;
```

```
00019 using System.Collections.Generic;
00020
00021 namespace VectSharp.PDF
00022 {
00023 /// <summary>
00024 /// Describes the appearance of a PDF annotation.
00025 /// </summary>
00026     public class AnnotationStyle
00027     {
00028 /// <summary>
00029 /// Thickness of the annotation border.
00030 /// </summary>
00031         public double BorderWidth { get; set; } = 1;
00032
00033 /// <summary>
00034 /// Dash style for the annotation border.
00035 /// </summary>
00036         public LineDash?  BorderDash { get; set; } = null;
00037
00038 /// <summary>
00039 /// Colour for the annotation border.
00040 /// </summary>
00041         public Colour BorderColour { get; set; } = Colours.Black;
00042
00043
00044 /// <summary>
00045 /// Create a new <see cref="AnnotationStyle"/>.
00046 /// </summary>
00047         public AnnotationStyle(double borderWidth = 1, LineDash?  borderDash = null, Colour?
      borderColour = default)
00048         {
00049             BorderWidth = borderWidth;
00050             BorderDash = borderDash;
00051             BorderColour = borderColour ??  Colours.Black;
00052         }
00053     }
00054
00055 /// <summary>
00056 /// Represents annotation style settings.
00057 /// </summary>
00058     public class AnnotationStyleCollection :  IDictionary<string, AnnotationStyle>
00059     {
00060 /// <summary>
00061 /// The default annotation style.
00062 /// </summary>
00063         public AnnotationStyle DefaultStyle { get; set; } = new AnnotationStyle();
00064
00065 /// <summary>
00066 /// Annotation style overrides for individual annotations.
00067 /// </summary>
00068         public Dictionary<string, AnnotationStyle> Styles { get; set; }
00069
00070 /// <inheritdoc/>
00071         public ICollection<string> Keys => ((IDictionary<string, AnnotationStyle>)Styles).Keys;
00072
00073 /// <inheritdoc/>
00074         public ICollection<AnnotationStyle> Values => ((IDictionary<string,
      AnnotationStyle>)Styles).Values;
00075
00076 /// <inheritdoc/>
00077         public int Count => ((ICollection<KeyValuePair<string, AnnotationStyle»)Styles).Count;
00078
00079 /// <inheritdoc/>
00080         public bool IsReadOnly => ((ICollection<KeyValuePair<string,
      AnnotationStyle»)Styles).IsReadOnly;
00081
00082 /// <inheritdoc/>
00083         public AnnotationStyle this[string key] { get => ((IDictionary<string,
      AnnotationStyle>)Styles)[key]; set => ((IDictionary<string, AnnotationStyle>)Styles)[key] = value; }
00084
00085 /// <summary>
00086 /// Create a new set of <see cref="AnnotationStyleCollection"/>.
00087 /// </summary>
00088 /// <param name="defaultStyle">The default <see cref="AnnotationStyle"/>.  If this is not provided or
      set to <see langword="null"/>, default values will be used.</param>
00089         public AnnotationStyleCollection(AnnotationStyle defaultStyle = null)
00090         {
00091             if (defaultStyle == null)
00092             {
00093                 defaultStyle = new AnnotationStyle();
00094             }
00095             this.DefaultStyle = defaultStyle;
00096
00097             this.Styles = new Dictionary<string, AnnotationStyle>();
00098         }
00099
00100 /// <inheritdoc/>
```

```
00101         public void Add(string key, AnnotationStyle value)
00102         {
00103             ((IDictionary<string, AnnotationStyle>)Styles).Add(key, value);
00104         }
00105 
00106 /// <inheritdoc/>
00107         public bool ContainsKey(string key)
00108         {
00109             return ((IDictionary<string, AnnotationStyle>)Styles).ContainsKey(key);
00110         }
00111 
00112 /// <inheritdoc/>
00113         public bool Remove(string key)
00114         {
00115             return ((IDictionary<string, AnnotationStyle>)Styles).Remove(key);
00116         }
00117 
00118 /// <inheritdoc/>
00119         public bool TryGetValue(string key, out AnnotationStyle value)
00120         {
00121             return ((IDictionary<string, AnnotationStyle>)Styles).TryGetValue(key, out value);
00122         }
00123 
00124 /// <inheritdoc/>
00125         public void Add(KeyValuePair<string, AnnotationStyle> item)
00126         {
00127             ((ICollection<KeyValuePair<string, AnnotationStyle>>)Styles).Add(item);
00128         }
00129 
00130 /// <inheritdoc/>
00131         public void Clear()
00132         {
00133             ((ICollection<KeyValuePair<string, AnnotationStyle>>)Styles).Clear();
00134         }
00135 
00136 /// <inheritdoc/>
00137         public bool Contains(KeyValuePair<string, AnnotationStyle> item)
00138         {
00139             return ((ICollection<KeyValuePair<string, AnnotationStyle>>)Styles).Contains(item);
00140         }
00141 
00142 /// <inheritdoc/>
00143         public void CopyTo(KeyValuePair<string, AnnotationStyle>[] array, int arrayIndex)
00144         {
00145             ((ICollection<KeyValuePair<string, AnnotationStyle>>)Styles).CopyTo(array, arrayIndex);
00146         }
00147 
00148 /// <inheritdoc/>
00149         public bool Remove(KeyValuePair<string, AnnotationStyle> item)
00150         {
00151             return ((ICollection<KeyValuePair<string, AnnotationStyle>>)Styles).Remove(item);
00152         }
00153 
00154 /// <inheritdoc/>
00155         public IEnumerator<KeyValuePair<string, AnnotationStyle>> GetEnumerator()
00156         {
00157             return ((IEnumerable<KeyValuePair<string, AnnotationStyle>>)Styles).GetEnumerator();
00158         }
00159 
00160 /// <inheritdoc/>
00161         IEnumerator IEnumerable.GetEnumerator()
00162         {
00163             return ((IEnumerable)Styles).GetEnumerator();
00164         }
00165     }
00166 }
```

8.28 Figures.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2024 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
```

```
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System.Collections.Generic;
00019 using VectSharp.PDF.OptionalContentGroups;
00020
00021 namespace VectSharp.PDF.Figures
00022 {
00023     internal enum SegmentType
00024     {
00025         Move, Line, CubicBezier, Close
00026     }
00027
00028     internal abstract class Segment
00029     {
00030         public abstract SegmentType Type { get; }
00031         public Point[] Points { get; protected set; }
00032
00033         public virtual Point Point
00034         {
00035             get
00036             {
00037                 return Points[Points.Length - 1];
00038             }
00039         }
00040
00041         public abstract Segment Clone();
00042     }
00043
00044     internal class MoveSegment :  Segment
00045     {
00046         public override SegmentType Type => SegmentType.Move;
00047
00048         public MoveSegment(Point p)
00049         {
00050             this.Points = new Point[] { p };
00051         }
00052
00053         public MoveSegment(double x, double y)
00054         {
00055             this.Points = new Point[] { new Point(x, y) };
00056         }
00057
00058         public override Segment Clone()
00059         {
00060             return new MoveSegment(this.Point);
00061         }
00062     }
00063
00064     internal class LineSegment :  Segment
00065     {
00066         public override SegmentType Type => SegmentType.Line;
00067
00068         public LineSegment(Point p)
00069         {
00070             this.Points = new Point[] { p };
00071         }
00072
00073         public LineSegment(double x, double y)
00074         {
00075             this.Points = new Point[] { new Point(x, y) };
00076         }
00077
00078         public override Segment Clone()
00079         {
00080             return new LineSegment(this.Point);
00081         }
00082     }
00083
00084     internal class CloseSegment :  Segment
00085     {
00086         public override SegmentType Type => SegmentType.Close;
00087
00088         public CloseSegment() { }
00089
00090         public override Segment Clone()
00091         {
00092             return new CloseSegment();
00093         }
00094     }
00095
00096     internal class CubicBezierSegment :  Segment
00097     {
00098         public override SegmentType Type => SegmentType.CubicBezier;
00099         public CubicBezierSegment(double x1, double y1, double x2, double y2, double x3, double y3)
00100         {
00101             Points = new Point[] { new Point(x1, y1), new Point(x2, y2), new Point(x3, y3) };
```

```
00102         }
00103
00104         public CubicBezierSegment(Point p1, Point p2, Point p3)
00105         {
00106             Points = new Point[] { p1, p2, p3 };
00107         }
00108
00109         public override Segment Clone()
00110         {
00111             return new CubicBezierSegment(Points[0], Points[1], Points[2]);
00112         }
00113     }
00114
00115     internal interface IFigure
00116     {
00117         Brush Fill { get; }
00118         Brush Stroke { get; }
00119         double LineWidth { get; }
00120         LineCaps LineCap { get; }
00121         LineJoins LineJoin { get; }
00122         LineDash LineDash { get; }
00123         bool IsClipping { get; }
00124         string Tag { get; }
00125         Rectangle GetBounds();
00126     }
00127
00128     internal class TransformFigure :  IFigure
00129     {
00130         public enum TransformTypes
00131         {
00132             Transform, Save, Restore
00133         }
00134
00135         public TransformTypes TransformType { get; }
00136
00137         public Brush Fill { get; }
00138         public Brush Stroke { get; }
00139
00140         public bool IsClipping { get; }
00141         public double LineWidth { get; }
00142
00143         public double[,] TransformationMatrix { get; }
00144
00145         public LineCaps LineCap { get; }
00146
00147         public LineJoins LineJoin { get; }
00148
00149         public LineDash LineDash { get; }
00150         public string Tag { get; }
00151
00152         public TransformFigure(TransformTypes type, double[,] transformationMatrix, string tag)
00153         {
00154             this.TransformType = type;
00155             this.TransformationMatrix = transformationMatrix;
00156             this.Tag = tag;
00157         }
00158
00159         public Rectangle GetBounds()
00160         {
00161             return Rectangle.NaN;
00162         }
00163     }
00164
00165     internal class RasterImageFigure :  IFigure
00166     {
00167
00168         public Brush Fill { get; }
00169         public Brush Stroke { get; }
00170
00171         public bool IsClipping { get; }
00172         public double LineWidth { get; }
00173
00174         public double[,] TransformationMatrix { get; }
00175
00176         public LineCaps LineCap { get; }
00177
00178         public LineJoins LineJoin { get; }
00179
00180         public LineDash LineDash { get; }
00181
00182         public RasterImage Image { get; }
00183
00184         public string Tag { get; }
00185
00186         public RasterImageFigure(RasterImage image, string tag)
00187         {
00188             this.Image = image;
```

```
00189             this.Tag = tag;
00190         }
00191
00192         public Rectangle GetBounds()
00193         {
00194             return new Rectangle(0, 0, 1, 1);
00195         }
00196     }
00197
00198     internal class PathFigure :  IFigure
00199     {
00200         public Brush Fill { get; }
00201         public FillRule FillRule { get; }
00202         public Brush Stroke { get; }
00203         public bool IsClipping { get; }
00204         public double LineWidth { get; }
00205
00206         public LineCaps LineCap { get; }
00207
00208         public LineJoins LineJoin { get; }
00209
00210         public LineDash LineDash { get; }
00211         public Segment[] Segments { get; }
00212
00213         public string Tag { get; }
00214
00215         private Rectangle Bounds { get; }
00216
00217         public PathFigure(IEnumerable<Segment> segments, Rectangle bounds, Brush fill, Brush stroke,
      double lineWidth, LineCaps lineCap, LineJoins lineJoin, LineDash lineDash, bool isClipping, FillRule
      fillRule, string tag)
00218         {
00219             List<Segment> segs = new List<Segment>();
00220
00221             foreach (Segment s in segments)
00222             {
00223                 segs.Add(s.Clone());
00224             }
00225
00226             this.Segments = segs.ToArray();
00227
00228             Fill = fill;
00229             Stroke = stroke;
00230             LineWidth = lineWidth;
00231             LineCap = lineCap;
00232             LineJoin = lineJoin;
00233             LineDash = lineDash;
00234             IsClipping = isClipping;
00235             this.Tag = tag;
00236             this.FillRule = fillRule;
00237
00238             if (stroke == null)
00239             {
00240                 this.Bounds = bounds;
00241             }
00242             else
00243             {
00244                 this.Bounds = new Rectangle(bounds.Location.X - lineWidth * 0.5, bounds.Location.Y -
      lineWidth * 0.5, bounds.Size.Width + lineWidth, bounds.Size.Height + lineWidth);
00245             }
00246         }
00247
00248         public Rectangle GetBounds()
00249         {
00250             return Bounds;
00251         }
00252     }
00253
00254     internal class TextFigure :  IFigure
00255     {
00256         public Brush Fill { get; }
00257         public Brush Stroke { get; }
00258         public double LineWidth { get; }
00259         public bool IsClipping { get; }
00260         public LineCaps LineCap { get; }
00261
00262         public LineJoins LineJoin { get; }
00263
00264         public LineDash LineDash { get; }
00265
00266         public string Text { get; }
00267
00268         public Font Font { get; }
00269
00270         public Point Position { get; }
00271
00272         public TextBaselines TextBaseline { get; }
```

```
00273
00274         public string Tag { get; }
00275
00276         public TextFigure(string text, Font font, Point position, TextBaselines textBaseline, Brush
      fill, Brush stroke, double lineWidth, LineCaps lineCap, LineJoins lineJoin, LineDash lineDash, string
      tag)
00277         {
00278             Text = text;
00279             Font = font;
00280             Position = position;
00281             TextBaseline = textBaseline;
00282
00283             Fill = fill;
00284             Stroke = stroke;
00285             LineWidth = lineWidth;
00286             LineCap = lineCap;
00287             LineJoin = lineJoin;
00288             LineDash = lineDash;
00289             this.Tag = tag;
00290         }
00291
00292         public Rectangle GetBounds()
00293         {
00294             Font.DetailedFontMetrics metrics = this.Font.MeasureTextAdvanced(this.Text);
00295
00296             switch (this.TextBaseline)
00297             {
00298                 case TextBaselines.Top:
00299                     return new Rectangle(this.Position, new Size(metrics.Width, metrics.Height));
00300                 case TextBaselines.Bottom:
00301                     return new Rectangle(this.Position.X, this.Position.Y - metrics.Height,
      metrics.Width, metrics.Height);
00302                 case TextBaselines.Middle:
00303                     return new Rectangle(this.Position.X, this.Position.Y - metrics.Height * 0.5,
      metrics.Width, metrics.Height);
00304                 case TextBaselines.Baseline:
00305                     return new Rectangle(this.Position.X, this.Position.Y - metrics.Top,
      metrics.Width, metrics.Height);
00306                 default:
00307                     throw new System.ArgumentOutOfRangeException(nameof(TextBaseline),
      this.TextBaseline, "Invalid text baseline!");
00308             }
00309         }
00310     }
00311
00312     internal class OptionalContentFigure :  IFigure
00313     {
00314         public enum OptionalContentType
00315         {
00316             Start, End
00317         }
00318
00319         public OptionalContentType FigureType { get; }
00320
00321         public Brush Fill { get; }
00322         public Brush Stroke { get; }
00323
00324         public bool IsClipping { get; }
00325         public double LineWidth { get; }
00326
00327         public OptionalContentGroupExpression VisibilityExpression { get; }
00328
00329         public LineCaps LineCap { get; }
00330
00331         public LineJoins LineJoin { get; }
00332
00333         public LineDash LineDash { get; }
00334         public string Tag { get; }
00335
00336         public OptionalContentFigure(OptionalContentType type, OptionalContentGroupExpression
      visibilityExpression)
00337         {
00338             this.FigureType = type;
00339             this.VisibilityExpression = visibilityExpression;
00340         }
00341
00342         public Rectangle GetBounds()
00343         {
00344             return Rectangle.NaN;
00345         }
00346     }
00347
00348 }
```

8.29 OptionalContentGroups.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2024 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections;
00020 using System.Collections.Generic;
00021 using System.Linq;
00022 using System.Text;
00023
00024 namespace VectSharp.PDF.OptionalContentGroups
00025 {
00026 /// <summary>
00027 /// Boolean operators for <see cref="OptionalContentGroup"/>s.
00028 /// </summary>
00029     public enum OptionalContentGroupBooleanOperation
00030     {
00031 /// <summary>
00032 /// Logical conjunction.
00033 /// </summary>
00034         And,
00035
00036 /// <summary>
00037 /// Logical disjunction.
00038 /// </summary>
00039         Or,
00040
00041 /// <summary>
00042 /// Logical negation.
00043 /// </summary>
00044         Not,
00045
00046 /// <summary>
00047 /// Logical identity.
00048 /// </summary>
00049         Identity
00050     }
00051
00052 /// <summary>
00053 /// Represents the result of a boolean operation on <see cref="OptionalContentGroup"/>s.
00054 /// </summary>
00055     public abstract class OptionalContentGroupExpression
00056     {
00057 /// <summary>
00058 /// The operation performed by the expression.
00059 /// </summary>
00060         public abstract OptionalContentGroupBooleanOperation Operation { get; }
00061
00062 /// <summary>
00063 /// The arguments of the expression.
00064 /// </summary>
00065         public virtual IReadOnlyList<OptionalContentGroupExpression> Arguments { get; }
00066
00067 /// <summary>
00068 /// Create a new <see cref="OptionalContentGroupExpression"/> with the specified <paramref
      name="arguments"/>.
00069 /// </summary>
00070 /// <param name="arguments">The arguments of the expression.</param>
00071         protected OptionalContentGroupExpression(params OptionalContentGroupExpression[] arguments)
00072         {
00073             this.Arguments = arguments;
00074         }
00075
00076 /// <summary>
00077 /// Negates an <see cref="OptionalContentGroupExpression"/>.
00078 /// </summary>
00079 /// <param name="ocge">The <see cref="OptionalContentGroupExpression"/> to be negated.</param>
00080         public static OptionalContentGroupNotExpression operator !(OptionalContentGroupExpression
      ocge)
00081         {
00082             return new OptionalContentGroupNotExpression(ocge);
00083         }
```

```
00084
00085 /// <summary>
00086 /// Used to enable short circuit operators.
00087 /// </summary>
00088 /// <param name="_">The expression.</param>
00089 /// <returns>Always <see langword="false"/>.</returns>
00090         public static bool operator false(OptionalContentGroupExpression _)
00091         {
00092             return false;
00093         }
00094
00095 /// <summary>
00096 /// Used to enable short circuit operators.
00097 /// </summary>
00098 /// <param name="_">The expression.</param>
00099 /// <returns>Always <see langword="false"/>.</returns>
00100         public static bool operator true(OptionalContentGroupExpression _)
00101         {
00102             return false;
00103         }
00104
00105 /// <summary>
00106 /// Performs a conjunction of two <see cref="OptionalContentGroupExpression"/>s.
00107 /// </summary>
00108 /// <param name="ocge1">The first expression.</param>
00109 /// <param name="ocge2">The second expression.</param>
00110 /// <returns>An <see cref="OptionalContentGroupExpression"/> representing the conjunction of the two
      expressions.</returns>
00111         public static OptionalContentGroupExpression operator &(OptionalContentGroupExpression ocge1,
      OptionalContentGroupExpression ocge2)
00112         {
00113             return new OptionalContentGroupAndExpression(ocge1, ocge2);
00114         }
00115
00116 /// <summary>
00117 /// Performs a disjunction of two <see cref="OptionalContentGroupExpression"/>s.
00118 /// </summary>
00119 /// <param name="ocge1">The first expression.</param>
00120 /// <param name="ocge2">The second expression.</param>
00121 /// <returns>An <see cref="OptionalContentGroupExpression"/> representing the disjunction of the two
      expressions.</returns>
00122         public static OptionalContentGroupExpression operator |(OptionalContentGroupExpression ocge1,
      OptionalContentGroupExpression ocge2)
00123         {
00124             return new OptionalContentGroupOrExpression(ocge1, ocge2);
00125         }
00126
00127         private string _cachedToString;
00128
00129 /// <inheritdoc/>
00130         public override string ToString()
00131         {
00132             if (_cachedToString == null)
00133             {
00134
00135                 StringBuilder sb = new StringBuilder();
00136                 sb.Append(Operation.ToString());
00137                 sb.Append("[");
00138                 for (int i = 0; i < Arguments.Count; i++)
00139                 {
00140                     sb.Append(Arguments[i].ToString());
00141                     if (i < Arguments.Count - 1)
00142                     {
00143                         sb.Append(",");
00144                     }
00145                 }
00146                 sb.Append("]");
00147                 _cachedToString = sb.ToString();
00148             }
00149
00150             return _cachedToString;
00151         }
00152     }
00153
00154 /// <summary>
00155 /// Represents an optional content group.
00156 /// </summary>
00157     public class OptionalContentGroup :  OptionalContentGroupExpression
00158     {
00159 /// <inheritdoc/>
00160         public override OptionalContentGroupBooleanOperation Operation =>
      OptionalContentGroupBooleanOperation.Identity;
00161
00162 /// <summary>
00163 /// The name of the <see cref="OptionalContentGroup"/>.
00164 /// </summary>
00165         public string Name { get; set; }
```

```
00166
00167 /// <summary>
00168 /// Possible intended uses of graphics in an <see cref="OptionalContentGroup"/>.
00169 /// </summary>
00170         public enum Intents
00171         {
00172 /// <summary>
00173 /// Intended for interactive use by document consumers.  This is probably what you want.
00174 /// </summary>
00175             View = 1,
00176
00177 /// <summary>
00178 /// Intended for structural organisation of artwork.
00179 /// </summary>
00180             Design = 2,
00181         }
00182
00183 /// <summary>
00184 /// The intended use of graphics in the <see cref="OptionalContentGroup"/>.
00185 /// </summary>
00186         public Intents Intent { get; set; }
00187
00188 /// <summary>
00189 /// The default state of the <see cref="OptionalContentGroup"/> when the document is opened.
00190 /// </summary>
00191         public bool DefaultState { get; set; }
00192
00193 /// <summary>
00194 /// The default state of the <see cref="OptionalContentGroup"/> when the document is opened for
      visualisation.  Set this to <see langword="null"/> to use the <see cref="DefaultState"/>.
00195 /// </summary>
00196         public bool?  DefaultViewState { get; set; }
00197
00198 /// <summary>
00199 /// The default state of the <see cref="OptionalContentGroup"/> when the document is opened for
      printing.  Set this to <see langword="null"/> to use the <see cref="DefaultState"/>.
00200 /// </summary>
00201         public bool?  DefaultPrintState { get; set; }
00202
00203 /// <summary>
00204 /// The default state of the <see cref="OptionalContentGroup"/> when the document is opened for
      exporting.  Set this to <see langword="null"/> to use the <see cref="DefaultState"/>.
00205 /// </summary>
00206         public bool?  DefaultExportState { get; set; }
00207
00208 /// <summary>
00209 /// A unique identifier for the <see cref="OptionalContentGroup"/>.
00210 /// </summary>
00211         private string Guid { get; } = System.Guid.NewGuid().ToString();
00212
00213 /// <summary>
00214 /// Create a new <see cref="OptionalContentGroup"/>.
00215 /// </summary>
00216 /// <param name="name">The name of the <see cref="OptionalContentGroup"/>.</param>
00217 /// <param name="defaultState">The default state of the <see cref="OptionalContentGroup"/> when the
      document is opened.</param>
00218 /// <param name="defaultViewState">The default state of the <see cref="OptionalContentGroup"/> when
      the document is opened for visualisation.  Set this to <see langword="null"/> to use the <see
      cref="DefaultState"/>.</param>
00219 /// <param name="defaultPrintState">The default state of the <see cref="OptionalContentGroup"/> when
      the document is opened for printing.  Set this to <see langword="null"/> to use the <see
      cref="DefaultState"/>.</param>
00220 /// <param name="defaultExportState">The default state of the <see cref="OptionalContentGroup"/> when
      the document is opened for exporting.  Set this to <see langword="null"/> to use the <see
      cref="DefaultState"/>.</param>
00221 /// <param name="intent">The intended use of graphics in the <see
      cref="OptionalContentGroup"/>.</param>
00222         public OptionalContentGroup(string name, bool defaultState = true, bool?  defaultViewState =
      null, bool?  defaultPrintState = null, bool?  defaultExportState = null, Intents intent =
      Intents.View)
00223         {
00224             this.Name = name;
00225             this.Intent = intent;
00226             this.DefaultState = defaultState;
00227             this.DefaultViewState = defaultViewState;
00228             this.DefaultPrintState = defaultPrintState;
00229             this.DefaultExportState = defaultExportState;
00230         }
00231
00232 /// <inheritdoc/>
00233         public override string ToString()
00234         {
00235             return this.Guid;
00236         }
00237
00238 /// <summary>
00239 /// Creates a link destination that sets the state of one or more <see cref="OptionalContentGroup"/>s.
```

```
00240 /// </summary>
00241 /// <param name="on"><see cref="OptionalContentGroup"/>s to turn on.</param>
00242 /// <param name="off"><see cref="OptionalContentGroup"/>s to turn off.</param>
00243 /// <param name="toggle"><see cref="OptionalContentGroup"/>s to toggle.</param>
00244 /// <returns>A string representation of an action that sets the specified optional content group
      states.  Use this within the <c>linkDestionations</c> parameter of the <see
      cref="PDFContextInterpreter.CreatePDFDocument"/> method.</returns>
00245 /// <exception cref="ArgumentException">Thrown if no <see cref="OptionalContentGroup"/>s are provided
      in any of the arguments.</exception>
00246         public static string CreateSetOCGLink(IReadOnlyList<OptionalContentGroup> on = null,
      IReadOnlyList<OptionalContentGroup> off = null, IReadOnlyList<OptionalContentGroup> toggle = null)
00247         {
00248             StringBuilder tbr = new StringBuilder("#@OCG:");
00249 
00250             bool any = false;
00251 
00252             if (on != null)
00253             {
00254                 for (int i = 0; i < on.Count; i++)
00255                 {
00256                     any = true;
00257                     tbr.Append(on[i].ToString());
00258                     tbr.Append(",on");
00259 
00260                     if (i < on.Count - 1 || off?.Count > 0 || toggle?.Count > 0)
00261                     {
00262                         tbr.Append(";");
00263                     }
00264                 }
00265             }
00266 
00267             if (off != null)
00268             {
00269                 for (int i = 0; i < off.Count; i++)
00270                 {
00271                     any = true;
00272                     tbr.Append(off[i].ToString());
00273                     tbr.Append(",off");
00274 
00275                     if (i < off.Count - 1 || toggle?.Count > 0)
00276                     {
00277                         tbr.Append(";");
00278                     }
00279                 }
00280             }
00281 
00282             if (toggle != null)
00283             {
00284                 for (int i = 0; i < toggle.Count; i++)
00285                 {
00286                     any = true;
00287                     tbr.Append(toggle[i].ToString());
00288                     tbr.Append(",toggle");
00289 
00290                     if (i < toggle.Count - 1)
00291                     {
00292                         tbr.Append(";");
00293                     }
00294                 }
00295             }
00296 
00297             if (!any)
00298             {
00299                 throw new ArgumentException("No optional content groups provided!");
00300             }
00301 
00302             return tbr.ToString();
00303         }
00304     }
00305 
00306 /// <summary>
00307 /// An expression that is true when its argument is false.
00308 /// </summary>
00309     public class OptionalContentGroupNotExpression :  OptionalContentGroupExpression
00310     {
00311 /// <inheritdoc/>
00312         public override OptionalContentGroupBooleanOperation Operation =>
      OptionalContentGroupBooleanOperation.Not;
00313 
00314 /// <summary>
00315 /// Create a new <see cref="OptionalContentGroupNotExpression"/>.
00316 /// </summary>
00317 /// <param name="expression">The <see cref="OptionalContentGroupNotExpression"/> that should be
      negated.</param>
00318         public OptionalContentGroupNotExpression(OptionalContentGroupExpression expression) :
      base(expression) { }
00319     }
```

```
00320
00321 /// <summary>
00322 /// An expression that is true when all its arguments are true.
00323 /// </summary>
00324     public class OptionalContentGroupAndExpression :  OptionalContentGroupExpression
00325     {
00326 /// <inheritdoc/>
00327         public override OptionalContentGroupBooleanOperation Operation =>
      OptionalContentGroupBooleanOperation.And;
00328
00329 /// <summary>
00330 /// Create a new <see cref="OptionalContentGroupAndExpression"/>.
00331 /// </summary>
00332 /// <param name="expression1">The first argument.</param>
00333 /// <param name="expression2">The second argument.</param>
00334 /// <param name="expressions">Additional arguments.</param>
00335         public OptionalContentGroupAndExpression(OptionalContentGroupExpression expression1,
      OptionalContentGroupExpression expression2, params OptionalContentGroupExpression[] expressions) :
      base(new OptionalContentGroupExpression[] { expression1, expression2 }.Concat(expressions).ToArray())
      { }
00336     }
00337
00338 /// <summary>
00339 /// An expression that is true when any of its arguments are true.
00340 /// </summary>
00341     public class OptionalContentGroupOrExpression :  OptionalContentGroupExpression
00342     {
00343 /// <inheritdoc/>
00344         public override OptionalContentGroupBooleanOperation Operation =>
      OptionalContentGroupBooleanOperation.Or;
00345
00346 /// <summary>
00347 /// Create a new <see cref="OptionalContentGroupOrExpression"/>.
00348 /// </summary>
00349 /// <param name="expression1">The first argument.</param>
00350 /// <param name="expression2">The second argument.</param>
00351 /// <param name="expressions">Additional arguments.</param>
00352         public OptionalContentGroupOrExpression(OptionalContentGroupExpression expression1,
      OptionalContentGroupExpression expression2, params OptionalContentGroupExpression[] expressions) :
      base(new OptionalContentGroupExpression[] { expression1, expression2 }.Concat(expressions).ToArray())
      { }
00353     }
00354
00355 /// <summary>
00356 /// A label for an <see cref="OptionalContentGroupTreeNode"/>.
00357 /// </summary>
00358     public class OptionalContentGroupTreeLabel
00359     {
00360 /// <summary>
00361 /// Types of label.
00362 /// </summary>
00363         public enum Type
00364         {
00365 /// <summary>
00366 /// A text string.
00367 /// </summary>
00368             String,
00369
00370 /// <summary>
00371 /// An <see cref="OptionalContentGroup"/>.
00372 /// </summary>
00373             OptionalContentGroup
00374         }
00375
00376 /// <summary>
00377 /// The type of label.
00378 /// </summary>
00379         public Type LabelType { get; }
00380
00381 /// <summary>
00382 /// If this label's <see cref="Type"/> is <see cref="Type.String"/>, the text of the label.
00383 /// </summary>
00384         public string LabelString { get; }
00385
00386 /// <summary>
00387 /// If this label's <see cref="Type"/> is <see cref="Type.OptionalContentGroup"/>, the <see
      cref="OptionalContentGroup"/> in the label.
00388 /// </summary>
00389         public OptionalContentGroup LabelOptionalContentGroup { get; }
00390
00391 /// <summary>
00392 /// Create a new <see cref="OptionalContentGroupTreeLabel"/> representing a text string.
00393 /// </summary>
00394 /// <param name="label">The text for the label.</param>
00395         public OptionalContentGroupTreeLabel(string label)
00396         {
00397             this.LabelType = Type.String;
```

```
00398             this.LabelString = label;
00399         }
00400
00401 /// <summary>
00402 /// Create a new <see cref="OptionalContentGroupTreeLabel"/> representing an <see
      cref="OptionalContentGroup"/>.
00403 /// </summary>
00404 /// <param name="label">The <see cref="OptionalContentGroup"/> in the label.</param>
00405         public OptionalContentGroupTreeLabel(OptionalContentGroup label)
00406         {
00407             this.LabelType = Type.OptionalContentGroup;
00408             this.LabelOptionalContentGroup = label;
00409         }
00410
00411 /// <summary>
00412 /// Convert a <see langword="string"/> into an <see cref="OptionalContentGroupTreeLabel"/>.
00413 /// </summary>
00414 /// <param name="label">The <see langword="string"/> to convert.</param>
00415         public static implicit operator OptionalContentGroupTreeLabel(string label) => new
      OptionalContentGroupTreeLabel(label);
00416
00417
00418 /// <summary>
00419 /// Convert a <see cref="OptionalContentGroup"/> into an <see cref="OptionalContentGroupTreeLabel"/>.
00420 /// </summary>
00421 /// <param name="label">The <see cref="OptionalContentGroup"/> to convert.</param>
00422         public static implicit operator OptionalContentGroupTreeLabel(OptionalContentGroup label) =>
      new OptionalContentGroupTreeLabel(label);
00423     }
00424
00425 /// <summary>
00426 /// Represents a node in an optional content group presentation tree.
00427 /// </summary>
00428     public class OptionalContentGroupTreeNode :  IList<OptionalContentGroupTreeNode>
00429     {
00430 /// <summary>
00431 /// Label for the node.  This can be a text string or an <see cref="OptionalContentGroup"/>.
00432 /// </summary>
00433         public OptionalContentGroupTreeLabel Label { get; set; }
00434
00435 /// <summary>
00436 /// Children for this node.
00437 /// </summary>
00438         public List<OptionalContentGroupTreeNode> Children { get; set; }
00439
00440 /// <inheritdoc/>
00441         public int Count => ((ICollection<OptionalContentGroupTreeNode>)Children).Count;
00442
00443 /// <inheritdoc/>
00444         public bool IsReadOnly => ((ICollection<OptionalContentGroupTreeNode>)Children).IsReadOnly;
00445
00446 /// <inheritdoc/>
00447         public OptionalContentGroupTreeNode this[int index] { get =>
      ((IList<OptionalContentGroupTreeNode>)Children)[index]; set =>
      ((IList<OptionalContentGroupTreeNode>)Children)[index] = value; }
00448
00449 /// <summary>
00450 /// Create a new <see cref="OptionalContentGroupTreeNode"/>.
00451 /// </summary>
00452 /// <param name="label">The label for the node.  This can be a text string or an <see
      cref="OptionalContentGroup"/>.</param>
00453         public OptionalContentGroupTreeNode(OptionalContentGroupTreeLabel label)
00454         {
00455             this.Label = label;
00456             this.Children = new List<OptionalContentGroupTreeNode>();
00457         }
00458 /// <summary>
00459 /// Convert an <see cref="OptionalContentGroupTreeLabel"/> into a <see
      cref="OptionalContentGroupTreeNode"/>.
00460 /// </summary>
00461 /// <param name="label">The node label.  This can be a text string or an <see
      cref="OptionalContentGroup"/>.</param>
00462         public static implicit operator OptionalContentGroupTreeNode(OptionalContentGroupTreeLabel
      label) => new OptionalContentGroupTreeNode(label);
00463
00464 /// <summary>
00465 /// Convert a <see langword="string"/> into a <see cref="OptionalContentGroupTreeNode"/>.
00466 /// </summary>
00467 /// <param name="label">The node label.</param>
00468         public static implicit operator OptionalContentGroupTreeNode(string label) => new
      OptionalContentGroupTreeLabel(label);
00469
00470 /// <summary>
00471 /// Convert an <see cref="OptionalContentGroup"/> into a <see cref="OptionalContentGroupTreeNode"/>.
00472 /// </summary>
00473 /// <param name="label">The node label.</param>
00474         public static implicit operator OptionalContentGroupTreeNode(OptionalContentGroup label) =>
```

```
      new OptionalContentGroupTreeLabel(label);
00475
00476 /// <inheritdoc/>
00477         public int IndexOf(OptionalContentGroupTreeNode item)
00478         {
00479             return ((IList<OptionalContentGroupTreeNode>)Children).IndexOf(item);
00480         }
00481
00482 /// <inheritdoc/>
00483         public void Insert(int index, OptionalContentGroupTreeNode item)
00484         {
00485             ((IList<OptionalContentGroupTreeNode>)Children).Insert(index, item);
00486         }
00487
00488 /// <inheritdoc/>
00489         public void RemoveAt(int index)
00490         {
00491             ((IList<OptionalContentGroupTreeNode>)Children).RemoveAt(index);
00492         }
00493
00494 /// <inheritdoc/>
00495         public void Add(OptionalContentGroupTreeNode item)
00496         {
00497             ((ICollection<OptionalContentGroupTreeNode>)Children).Add(item);
00498         }
00499
00500 /// <inheritdoc/>
00501         public void Clear()
00502         {
00503             ((ICollection<OptionalContentGroupTreeNode>)Children).Clear();
00504         }
00505
00506 /// <inheritdoc/>
00507         public bool Contains(OptionalContentGroupTreeNode item)
00508         {
00509             return ((ICollection<OptionalContentGroupTreeNode>)Children).Contains(item);
00510         }
00511
00512 /// <inheritdoc/>
00513         public void CopyTo(OptionalContentGroupTreeNode[] array, int arrayIndex)
00514         {
00515             ((ICollection<OptionalContentGroupTreeNode>)Children).CopyTo(array, arrayIndex);
00516         }
00517
00518 /// <inheritdoc/>
00519         public bool Remove(OptionalContentGroupTreeNode item)
00520         {
00521             return ((ICollection<OptionalContentGroupTreeNode>)Children).Remove(item);
00522         }
00523
00524 /// <inheritdoc/>
00525         public IEnumerator<OptionalContentGroupTreeNode> GetEnumerator()
00526         {
00527             return ((IEnumerable<OptionalContentGroupTreeNode>)Children).GetEnumerator();
00528         }
00529
00530 /// <inheritdoc/>
00531         IEnumerator IEnumerable.GetEnumerator()
00532         {
00533             return ((IEnumerable)Children).GetEnumerator();
00534         }
00535     }
00536
00537 /// <summary>
00538 /// Represents the settings for optional content groups (sometimes referred to as layers) in a PDF
      document.
00539 /// </summary>
00540     public class OptionalContentGroupSettings
00541     {
00542 /// <summary>
00543 /// A dictionary associating graphics tags to the corresponding visibility criteria.
00544 /// </summary>
00545         public Dictionary<string, OptionalContentGroupExpression> Groups { get; set; }
00546
00547 /// <summary>
00548 /// A tree structure used by PDF viewers to present the optional content groups to the user.  If this
      is set to <see langword="null"/> (the default), a flat tree (i.e.  a list of all content groups) will
      be generated automatically.  If this is empty but not <see langword="null"/>, no content groups are
      presented.
00549 /// </summary>
00550         public List<OptionalContentGroupTreeNode> OptionalContentGroupTree { get; set; }
00551
00552 /// <summary>
00553 /// Radio button groups.  For each group, at most one <see cref="OptionalContentGroup"/> can be
      enabled at the same time.
00554 /// </summary>
00555         public List<List<OptionalContentGroup» RadioButtonGroups { get; set; }
```

```
00556
00557 /// <summary>
00558 /// Create a new <see cref="OptionalContentGroupSettings"/> object.
00559 /// </summary>
00560         public OptionalContentGroupSettings()
00561         {
00562             this.Groups = new Dictionary<string, OptionalContentGroupExpression>();
00563             this.OptionalContentGroupTree = null;
00564             this.RadioButtonGroups = new List<List<OptionalContentGroup»();
00565         }
00566     }
00567 }
```

8.30 OutlineTree.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2024 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System.Collections;
00019 using System.Collections.Generic;
00020
00021 namespace VectSharp.PDF
00022 {
00023 /// <summary>
00024 /// Represents an outline tree (table of contents) for a PDF document.
00025 /// </summary>
00026     public class OutlineTree :  IList<OutlineTreeNode>
00027     {
00028 /// <summary>
00029 /// The top-level items in the outline.
00030 /// </summary>
00031         public List<OutlineTreeNode> TopLevelItems { get; }
00032
00033 /// <inheritdoc/>
00034         public int Count => ((ICollection<OutlineTreeNode>)TopLevelItems).Count;
00035
00036 /// <inheritdoc/>
00037         public bool IsReadOnly => ((ICollection<OutlineTreeNode>)TopLevelItems).IsReadOnly;
00038
00039 /// <inheritdoc/>
00040         public OutlineTreeNode this[int index] { get =>
      ((IList<OutlineTreeNode>)TopLevelItems)[index]; set => ((IList<OutlineTreeNode>)TopLevelItems)[index]
      = value; }
00041
00042 /// <summary>
00043 /// Create a new <see cref="OutlineTree"/>.
00044 /// </summary>
00045         public OutlineTree()
00046         {
00047             this.TopLevelItems = new List<OutlineTreeNode>();
00048         }
00049
00050 /// <summary>
00051 /// Creates an <see cref="OutlineTree"/> from a list of headings, such as the one produced by
      VectSharp.Markdown.
00052 /// </summary>
00053 /// <param name="headings">The list of headings.  Each element specifies a heading level (starting
      from 1), the text for the heading, and a tag for the corresponding object in the document.</param>
00054 /// <param name="topLevel">The heading level for top-level headings.  Headings above this level will
      not be included in the outline tree.  Setting this to a value higher than 1 is useful e.g.  if the
      document has a title included as a level 1 heading.</param>
00055 /// <returns>An <see cref="OutlineTree"/> representation of the heading list.</returns>
00056         public static OutlineTree CreateFromHeadings(List<(int level, string heading, string tag)>
      headings, int topLevel = 1)
00057         {
00058             OutlineTree tbr = new OutlineTree();
00059
00060             Stack<(IList<OutlineTreeNode>, int)> parents = new Stack<(IList<OutlineTreeNode>, int)>();
00061             parents.Push((tbr, 0));
```

```
00062
00063             for (int i = 0; i < headings.Count; i++)
00064             {
00065                 if (headings[i].level >= topLevel)
00066                 {
00067                     int lev = headings[i].level - topLevel + 1;
00068
00069                     OutlineTreeNode item = new OutlineTreeNode(headings[i].heading, headings[i].tag);
00070
00071                     while (lev <= parents.Peek().Item2)
00072                     {
00073                         parents.Pop();
00074                     }
00075
00076                     parents.Peek().Item1.Add(item);
00077                     parents.Push((item, lev));
00078                 }
00079             }
00080
00081             return tbr;
00082         }
00083
00084
00085 /// <inheritdoc/>
00086         public int IndexOf(OutlineTreeNode item)
00087         {
00088             return ((IList<OutlineTreeNode>)TopLevelItems).IndexOf(item);
00089         }
00090
00091 /// <inheritdoc/>
00092         public void Insert(int index, OutlineTreeNode item)
00093         {
00094             ((IList<OutlineTreeNode>)TopLevelItems).Insert(index, item);
00095         }
00096
00097 /// <inheritdoc/>
00098         public void RemoveAt(int index)
00099         {
00100             ((IList<OutlineTreeNode>)TopLevelItems).RemoveAt(index);
00101         }
00102
00103 /// <inheritdoc/>
00104         public void Add(OutlineTreeNode item)
00105         {
00106             ((ICollection<OutlineTreeNode>)TopLevelItems).Add(item);
00107         }
00108
00109 /// <inheritdoc/>
00110         public void Clear()
00111         {
00112             ((ICollection<OutlineTreeNode>)TopLevelItems).Clear();
00113         }
00114
00115 /// <inheritdoc/>
00116         public bool Contains(OutlineTreeNode item)
00117         {
00118             return ((ICollection<OutlineTreeNode>)TopLevelItems).Contains(item);
00119         }
00120
00121 /// <inheritdoc/>
00122         public void CopyTo(OutlineTreeNode[] array, int arrayIndex)
00123         {
00124             ((ICollection<OutlineTreeNode>)TopLevelItems).CopyTo(array, arrayIndex);
00125         }
00126
00127 /// <inheritdoc/>
00128         public bool Remove(OutlineTreeNode item)
00129         {
00130             return ((ICollection<OutlineTreeNode>)TopLevelItems).Remove(item);
00131         }
00132
00133 /// <inheritdoc/>
00134         public IEnumerator<OutlineTreeNode> GetEnumerator()
00135         {
00136             return ((IEnumerable<OutlineTreeNode>)TopLevelItems).GetEnumerator();
00137         }
00138
00139         IEnumerator IEnumerable.GetEnumerator()
00140         {
00141             return ((IEnumerable)TopLevelItems).GetEnumerator();
00142         }
00143     }
00144
00145 /// <summary>
00146 /// An item in an outline tree.
00147 /// </summary>
00148     public class OutlineTreeNode :  IList<OutlineTreeNode>
```

```
00149     {
00150 /// <summary>
00151 /// Children of this item (can be empty).
00152 /// </summary>
00153         public List<OutlineTreeNode> Children { get; }
00154 
00155 /// <summary>
00156 /// Text for this item.
00157 /// </summary>
00158         public string Title { get; set; }
00159 
00160 /// <summary>
00161 /// Tag of the graphics item that should be focused when the user clicks on the outline item.
00162 /// </summary>
00163         public string DestinationTag { get; set; }
00164 
00165 /// <summary>
00166 /// Colour for the outline item (the alpha component is ignored).
00167 /// </summary>
00168         public Colour Colour { get; set; }
00169 
00170 /// <summary>
00171 /// Indicates whether the outline item should be highlighted in bold or not.
00172 /// </summary>
00173         public bool Bold { get; set; }
00174 
00175 /// <summary>
00176 /// Indicates whether the outline item should be highlighted in italics or not.
00177 /// </summary>
00178         public bool Italic { get; set; }
00179 
00180 /// <inheritdoc/>
00181         public int Count => ((ICollection<OutlineTreeNode>)Children).Count;
00182 
00183 /// <inheritdoc/>
00184         public bool IsReadOnly => ((ICollection<OutlineTreeNode>)Children).IsReadOnly;
00185 
00186 /// <inheritdoc/>
00187         public OutlineTreeNode this[int index] { get => ((IList<OutlineTreeNode>)Children)[index]; set
      => ((IList<OutlineTreeNode>)Children)[index] = value; }
00188 
00189 /// <summary>
00190 /// Create a new <see cref="OutlineTreeNode"/>.
00191 /// </summary>
00192 /// <param name="title">The text for the new outline item.</param>
00193 /// <param name="destinationTag">Tag of the graphics item that should be focused when the user clicks
      on the outline item.</param>
00194 /// <param name="colour">The colour for the outline item.</param>
00195 /// <param name="bold">Whether the outline item should be highlighted in bold or not.</param>
00196 /// <param name="italic">Whether the outline item should be highlighted in italics or not.</param>
00197         public OutlineTreeNode(string title, string destinationTag = null, Colour colour = default,
      bool bold = false, bool italic = false)
00198         {
00199             this.Title = title;
00200             this.Children = new List<OutlineTreeNode>();
00201             this.DestinationTag = destinationTag;
00202             Colour = colour;
00203             Bold = bold;
00204             Italic = italic;
00205         }
00206 
00207 /// <inheritdoc/>
00208         public int IndexOf(OutlineTreeNode item)
00209         {
00210             return ((IList<OutlineTreeNode>)Children).IndexOf(item);
00211         }
00212 
00213 /// <inheritdoc/>
00214         public void Insert(int index, OutlineTreeNode item)
00215         {
00216             ((IList<OutlineTreeNode>)Children).Insert(index, item);
00217         }
00218 
00219 /// <inheritdoc/>
00220         public void RemoveAt(int index)
00221         {
00222             ((IList<OutlineTreeNode>)Children).RemoveAt(index);
00223         }
00224 
00225 /// <inheritdoc/>
00226         public void Add(OutlineTreeNode item)
00227         {
00228             ((ICollection<OutlineTreeNode>)Children).Add(item);
00229         }
00230 
00231 /// <inheritdoc/>
00232         public void Clear()
```

```
00233         {
00234             ((ICollection<OutlineTreeNode>)Children).Clear();
00235         }
00236 
00237 /// <inheritdoc/>
00238         public bool Contains(OutlineTreeNode item)
00239         {
00240             return ((ICollection<OutlineTreeNode>)Children).Contains(item);
00241         }
00242 
00243 /// <inheritdoc/>
00244         public void CopyTo(OutlineTreeNode[] array, int arrayIndex)
00245         {
00246             ((ICollection<OutlineTreeNode>)Children).CopyTo(array, arrayIndex);
00247         }
00248 
00249 /// <inheritdoc/>
00250         public bool Remove(OutlineTreeNode item)
00251         {
00252             return ((ICollection<OutlineTreeNode>)Children).Remove(item);
00253         }
00254 
00255 /// <inheritdoc/>
00256         public IEnumerator<OutlineTreeNode> GetEnumerator()
00257         {
00258             return ((IEnumerable<OutlineTreeNode>)Children).GetEnumerator();
00259         }
00260 
00261         IEnumerator IEnumerable.GetEnumerator()
00262         {
00263             return ((IEnumerable)Children).GetEnumerator();
00264         }
00265     }
00266 }
```

8.31 PDFContext.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2024 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021 using VectSharp.Filters;
00022 using VectSharp.PDF.Figures;
00023 using VectSharp.PDF.OptionalContentGroups;
00024 
00025 namespace VectSharp.PDF
00026 {
00027     internal class PDFContext :  IGraphicsContext
00028     {
00029         public string Tag { get; set; }
00030         public double Width { get; }
00031         public double Height { get; }
00032 
00033         public Dictionary<string, (FontFamily, HashSet<char>)> FontFamilies;
00034 
00035         public Dictionary<string, RasterImage> Images;
00036 
00037         public HashSet<double> Alphas;
00038 
00039         private List<Figures.Segment> _currentFigure;
00040 
00041         internal List<IFigure> _figures;
00042 
00043         private Brush _strokeStyle;
00044         private Brush _fillStyle;
00045         private LineDash _lineDash;
00046 
```

```
00047         private readonly bool _textToPaths;
00048
00049         private readonly PDFContextInterpreter.FilterOption _filterOption;
00050
00051         private OptionalContentGroupExpression _currentVisibilityExpression = null;
00052         private readonly Dictionary<string, OptionalContentGroupExpression> _ocgVisibilityExpressions;
00053         public readonly Dictionary<string, OptionalContentGroupExpression> VisibilityExpressions;
00054
00055         public PDFContext(double width, double height, Colour background, Dictionary<string,
      (FontFamily, HashSet<char>)> fontFamilies, Dictionary<string, RasterImage> images, HashSet<double>
      alphas, bool textToPaths, PDFContextInterpreter.FilterOption filterOption, Dictionary<string,
      OptionalContentGroupExpression> ocgVisibilityExpressions, Dictionary<string,
      OptionalContentGroupExpression> visibilityExpressions)
00056         {
00057             this.Width = width;
00058             this.Height = height;
00059
00060             this._ocgVisibilityExpressions = ocgVisibilityExpressions;
00061             this.VisibilityExpressions = visibilityExpressions;
00062
00063             _currentFigure = new List<Figures.Segment>();
00064             _figures = new List<IFigure>();
00065
00066             _strokeStyle = Colour.FromRgb(0, 0, 0);
00067             _fillStyle = Colour.FromRgb(0, 0, 0);
00068             LineWidth = 1;
00069
00070             LineCap = LineCaps.Butt;
00071             LineJoin = LineJoins.Miter;
00072             _lineDash = new LineDash(0, 0, 0);
00073
00074             _textToPaths = textToPaths;
00075
00076             Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica),
      12);
00077
00078             TextBaseline = TextBaselines.Top;
00079
00080             this.Translate(0, height);
00081             this.Scale(1, -1);
00082
00083             this.FontFamilies = fontFamilies;
00084             this.Images = images;
00085             this.Alphas = alphas;
00086
00087             this.Rectangle(0, 0, width, height);
00088             this.SetFillStyle(background);
00089             this.Fill(FillRule.NonZeroWinding);
00090
00091             this.SetFillStyle(Colour.FromRgb(0, 0, 0));
00092
00093             this._filterOption = filterOption;
00094         }
00095
00096         public void Finish()
00097         {
00098             if (_currentVisibilityExpression != null)
00099             {
00100                 _figures.Add(new OptionalContentFigure(OptionalContentFigure.OptionalContentType.End,
      _currentVisibilityExpression));
00101             }
00102         }
00103
00104
00105         public void MoveTo(double x, double y)
00106         {
00107             _currentFigure.Add(new MoveSegment(x, y));
00108         }
00109
00110         public void LineTo(double x, double y)
00111         {
00112             _currentFigure.Add(new LineSegment(x, y));
00113         }
00114
00115         public void Close()
00116         {
00117             _currentFigure.Add(new CloseSegment());
00118         }
00119
00120         public void Rectangle(double x0, double y0, double width, double height)
00121         {
00122             MoveTo(x0, y0);
00123             LineTo(x0 + width, y0);
00124             LineTo(x0 + width, y0 + height);
00125             LineTo(x0, y0 + height);
00126             Close();
00127         }
```

```
00128         public void SetStrokeStyle((int r, int g, int b, double a) style)
00129         {
00130             _strokeStyle = Colour.FromRgba(style.r, style.g, style.b, style.a);
00131         }
00132 
00133         public void SetStrokeStyle(Brush style)
00134         {
00135             _strokeStyle = style;
00136         }
00137 
00138         public void SetFillStyle((int r, int g, int b, double a) style)
00139         {
00140             _fillStyle = Colour.FromRgba(style.r, style.g, style.b, style.a);
00141         }
00142 
00143         public void SetFillStyle(Brush style)
00144         {
00145             _fillStyle = style;
00146         }
00147 
00148 
00149         public Brush FillStyle { get { return _fillStyle; } }
00150         public Brush StrokeStyle { get { return _strokeStyle; } }
00151 
00152         public double LineWidth { get; set; }
00153 
00154         public LineCaps LineCap { get; set; }
00155         public LineJoins LineJoin { get; set; }
00156 
00157         internal static bool IsCompatible(Brush brush)
00158         {
00159             if (brush is SolidColourBrush)
00160             {
00161                 return true;
00162             }
00163             else if (brush is GradientBrush gradient)
00164             {
00165                 foreach (GradientStop stop in gradient.GradientStops)
00166                 {
00167                     if (stop.Colour.A != 1)
00168                     {
00169                         return false;
00170                     }
00171                 }
00172                 return true;
00173             }
00174             else
00175             {
00176                 return false;
00177             }
00178         }
00179 
00180         internal static Brush GetAlphaBrush(Brush brush)
00181         {
00182             if (brush is SolidColourBrush)
00183             {
00184                 return brush;
00185             }
00186             else if (brush is LinearGradientBrush linear)
00187             {
00188                 return new LinearGradientBrush(linear.StartPoint, linear.EndPoint, from el in
      linear.GradientStops select new GradientStop(Colour.FromRgb(el.Colour.A, el.Colour.A, el.Colour.A),
      el.Offset));
00189             }
00190             else if (brush is RadialGradientBrush radial)
00191             {
00192                 return new RadialGradientBrush(radial.FocalPoint, radial.Centre, radial.Radius, from
      el in radial.GradientStops select new GradientStop(Colour.FromRgb(el.Colour.A, el.Colour.A,
      el.Colour.A), el.Offset));
00193             }
00194             else
00195             {
00196                 return null;
00197             }
00198         }
00199 
00200         private static GraphicsPath GetGraphicsPath(IEnumerable<Figures.Segment> segments)
00201         {
00202             GraphicsPath tbr = new GraphicsPath();
00203 
00204             foreach (Figures.Segment seg in segments)
00205             {
00206                 switch (seg.Type)
00207                 {
00208                     case Figures.SegmentType.Close:
00209                         tbr.Close();
00210                         break;
```

```
00211                     case Figures.SegmentType.Move:
00212                         tbr.MoveTo(seg.Point);
00213                         break;
00214                     case Figures.SegmentType.Line:
00215                         tbr.LineTo(seg.Point);
00216                         break;
00217                     case Figures.SegmentType.CubicBezier:
00218                         tbr.CubicBezierTo(seg.Points[0], seg.Points[1], seg.Points[2]);
00219                         break;
00220                 }
00221             }
00222 
00223             return tbr;
00224         }
00225 
00226         public void Fill(FillRule fillRule)
00227         {
00228             OptionalContentGroupExpression ocge = null;
00229 
00230             if (_ocgVisibilityExpressions != null && this.Tag != null)
00231             {
00232                 _ocgVisibilityExpressions.TryGetValue(this.Tag, out ocge);
00233             }
00234 
00235             if (ocge != _currentVisibilityExpression)
00236             {
00237                 if (_currentVisibilityExpression != null)
00238                 {
00239                     _figures.Add(new
    OptionalContentFigure(OptionalContentFigure.OptionalContentType.End, _currentVisibilityExpression));
00240                 }
00241 
00242                 if (ocge != null)
00243                 {
00244                     _figures.Add(new
    OptionalContentFigure(OptionalContentFigure.OptionalContentType.Start, ocge));
00245                     this.VisibilityExpressions[ocge.ToString()] = ocge;
00246                 }
00247 
00248                 _currentVisibilityExpression = ocge;
00249             }
00250 
00251             if (IsCompatible(_fillStyle))
00252             {
00253                 _figures.Add(new PathFigure(_currentFigure, VectSharp.Rectangle.NaN, _fillStyle, null,
    0, LineCaps.Butt, LineJoins.Bevel, new LineDash(0, 0, 0), false, fillRule, this.Tag));
00254             }
00255             else
00256             {
00257                 _figures.Add(new PathFigure(_currentFigure,
    GetGraphicsPath(_currentFigure).GetBounds(), _fillStyle, null, 0, LineCaps.Butt, LineJoins.Bevel, new
    LineDash(0, 0, 0), false, fillRule, this.Tag));
00258             }
00259 
00260             if (this.FillStyle is SolidColourBrush solidColourBrush)
00261             {
00262                 this.Alphas.Add(solidColourBrush.A);
00263             }
00264             else if (this.FillStyle is GradientBrush gradientBrush)
00265             {
00266                 foreach (GradientStop stop in gradientBrush.GradientStops)
00267                 {
00268                     this.Alphas.Add(stop.Colour.A);
00269                 }
00270             }
00271 
00272             _currentFigure = new List<Figures.Segment>();
00273         }
00274 
00275         public void Stroke()
00276         {
00277             OptionalContentGroupExpression ocge = null;
00278 
00279             if (_ocgVisibilityExpressions != null && this.Tag != null)
00280             {
00281                 _ocgVisibilityExpressions.TryGetValue(this.Tag, out ocge);
00282             }
00283 
00284             if (ocge != _currentVisibilityExpression)
00285             {
00286                 if (_currentVisibilityExpression != null)
00287                 {
00288                     _figures.Add(new
    OptionalContentFigure(OptionalContentFigure.OptionalContentType.End, _currentVisibilityExpression));
00289                 }
00290 
00291                 if (ocge != null)
```

```
00292                 {
00293                     _figures.Add(new
    OptionalContentFigure(OptionalContentFigure.OptionalContentType.Start, ocge));
00294                     this.VisibilityExpressions[ocge.ToString()] = ocge;
00295                 }
00296 
00297                 _currentVisibilityExpression = ocge;
00298             }
00299 
00300             if (IsCompatible(_strokeStyle))
00301             {
00302                 _figures.Add(new PathFigure(_currentFigure, VectSharp.Rectangle.NaN, null,
    _strokeStyle, LineWidth, LineCap, LineJoin, _lineDash, false, FillRule.NonZeroWinding, this.Tag));
00303             }
00304             else
00305             {
00306                 _figures.Add(new PathFigure(_currentFigure,
    GetGraphicsPath(_currentFigure).GetBounds(), null, _strokeStyle, LineWidth, LineCap, LineJoin,
    _lineDash, false, FillRule.NonZeroWinding, this.Tag));
00307             }
00308 
00309             if (this.StrokeStyle is SolidColourBrush solidColourBrush)
00310             {
00311                 this.Alphas.Add(solidColourBrush.A);
00312             }
00313             else if (this.StrokeStyle is GradientBrush gradientBrush)
00314             {
00315                 foreach (GradientStop stop in gradientBrush.GradientStops)
00316                 {
00317                     this.Alphas.Add(stop.Colour.A);
00318                 }
00319             }
00320 
00321             _currentFigure = new List<Figures.Segment>();
00322         }
00323 
00324         public void SetClippingPath()
00325         {
00326             _figures.Add(new PathFigure(_currentFigure, VectSharp.Rectangle.NaN, null, null, 0,
    LineCaps.Butt, LineJoins.Bevel, new LineDash(0, 0, 0), true, FillRule.NonZeroWinding, this.Tag));
00327             _currentFigure = new List<Figures.Segment>();
00328         }
00329 
00330         public void CubicBezierTo(double x1, double y1, double x2, double y2, double x3, double y3)
00331         {
00332             _currentFigure.Add(new CubicBezierSegment(x1, y1, x2, y2, x3, y3));
00333         }
00334 
00335         public void SetLineDash(LineDash dash)
00336         {
00337             _lineDash = dash;
00338         }
00339 
00340         public Font Font { get; set; }
00341 
00342         private void PathText(string text, double x, double y)
00343         {
00344             GraphicsPath textPath = new GraphicsPath().AddText(x, y, text, Font, TextBaseline);
00345 
00346             for (int j = 0; j < textPath.Segments.Count; j++)
00347             {
00348                 switch (textPath.Segments[j].Type)
00349                 {
00350                     case VectSharp.SegmentType.Move:
00351                         this.MoveTo(textPath.Segments[j].Point.X, textPath.Segments[j].Point.Y);
00352                         break;
00353                     case VectSharp.SegmentType.Line:
00354                         this.LineTo(textPath.Segments[j].Point.X, textPath.Segments[j].Point.Y);
00355                         break;
00356                     case VectSharp.SegmentType.CubicBezier:
00357                         this.CubicBezierTo(textPath.Segments[j].Points[0].X,
    textPath.Segments[j].Points[0].Y, textPath.Segments[j].Points[1].X, textPath.Segments[j].Points[1].Y,
    textPath.Segments[j].Points[2].X, textPath.Segments[j].Points[2].Y);
00358                         break;
00359                     case VectSharp.SegmentType.Close:
00360                         this.Close();
00361                         break;
00362                 }
00363             }
00364         }
00365 
00366         public void FillText(string text, double x, double y)
00367         {
00368             if (!_textToPaths)
00369             {
00370                 OptionalContentGroupExpression ocge = null;
00371 
```

```
00372                 if (_ocgVisibilityExpressions != null && this.Tag != null)
00373                 {
00374                     _ocgVisibilityExpressions.TryGetValue(this.Tag, out ocge);
00375                 }
00376 
00377                 if (ocge != _currentVisibilityExpression)
00378                 {
00379                     if (_currentVisibilityExpression != null)
00380                     {
00381                         _figures.Add(new
      OptionalContentFigure(OptionalContentFigure.OptionalContentType.End, _currentVisibilityExpression));
00382                     }
00383 
00384                     if (ocge != null)
00385                     {
00386                         _figures.Add(new
      OptionalContentFigure(OptionalContentFigure.OptionalContentType.Start, ocge));
00387                         this.VisibilityExpressions[ocge.ToString()] = ocge;
00388                     }
00389 
00390                     _currentVisibilityExpression = ocge;
00391                 }
00392 
00393                 if (!FontFamilies.TryGetValue(this.Font.FontFamily.FileName, out (FontFamily,
      HashSet<char>) usedChars))
00394                 {
00395                     usedChars = (this.Font.FontFamily, new HashSet<char>());
00396                     FontFamilies[this.Font.FontFamily.FileName] = usedChars;
00397                 }
00398 
00399                 foreach (char c in text)
00400                 {
00401                     usedChars.Item2.Add(c);
00402                 }
00403 
00404                 _figures.Add(new TextFigure(text, Font, new Point(x, y), TextBaseline, _fillStyle,
      null, 0, LineCaps.Butt, LineJoins.Miter, new LineDash(0, 0, 0), this.Tag));
00405             }
00406             else
00407             {
00408                 PathText(text, x, y);
00409                 Fill(FillRule.NonZeroWinding);
00410             }
00411 
00412             if (this.FillStyle is SolidColourBrush solidColourBrush)
00413             {
00414                 this.Alphas.Add(solidColourBrush.A);
00415             }
00416             else if (this.FillStyle is GradientBrush gradientBrush)
00417             {
00418                 foreach (GradientStop stop in gradientBrush.GradientStops)
00419                 {
00420                     this.Alphas.Add(stop.Colour.A);
00421                 }
00422             }
00423         }
00424 
00425         public TextBaselines TextBaseline { get; set; }
00426 
00427         public void Restore()
00428         {
00429             OptionalContentGroupExpression ocge = null;
00430 
00431             if (_ocgVisibilityExpressions != null && this.Tag != null)
00432             {
00433                 _ocgVisibilityExpressions.TryGetValue(this.Tag, out ocge);
00434             }
00435 
00436             if (ocge != _currentVisibilityExpression)
00437             {
00438                 if (_currentVisibilityExpression != null)
00439                 {
00440                     _figures.Add(new
      OptionalContentFigure(OptionalContentFigure.OptionalContentType.End, _currentVisibilityExpression));
00441                 }
00442 
00443                 if (ocge != null)
00444                 {
00445                     _figures.Add(new
      OptionalContentFigure(OptionalContentFigure.OptionalContentType.Start, ocge));
00446                     this.VisibilityExpressions[ocge.ToString()] = ocge;
00447                 }
00448 
00449                 _currentVisibilityExpression = ocge;
00450             }
00451 
00452             _figures.Add(new TransformFigure(TransformFigure.TransformTypes.Restore, null, this.Tag));
```

```
00453         }
00454
00455         public void Rotate(double angle)
00456         {
00457             OptionalContentGroupExpression ocge = null;
00458
00459             if (_ocgVisibilityExpressions != null && this.Tag != null)
00460             {
00461                 _ocgVisibilityExpressions.TryGetValue(this.Tag, out ocge);
00462             }
00463
00464             if (ocge != _currentVisibilityExpression)
00465             {
00466                 if (_currentVisibilityExpression != null)
00467                 {
00468                     _figures.Add(new
      OptionalContentFigure(OptionalContentFigure.OptionalContentType.End, _currentVisibilityExpression));
00469                 }
00470
00471                 if (ocge != null)
00472                 {
00473                     _figures.Add(new
      OptionalContentFigure(OptionalContentFigure.OptionalContentType.Start, ocge));
00474                     this.VisibilityExpressions[ocge.ToString()] = ocge;
00475                 }
00476
00477                 _currentVisibilityExpression = ocge;
00478             }
00479
00480             _figures.Add(new TransformFigure(TransformFigure.TransformTypes.Transform, new double[,] {
      { Math.Cos(angle), Math.Sin(angle), 0 }, { -Math.Sin(angle), Math.Cos(angle), 0 }, { 0, 0, 1 } },
      this.Tag));
00481         }
00482
00483         public void Save()
00484         {
00485             OptionalContentGroupExpression ocge = null;
00486
00487             if (_ocgVisibilityExpressions != null && this.Tag != null)
00488             {
00489                 _ocgVisibilityExpressions.TryGetValue(this.Tag, out ocge);
00490             }
00491
00492             if (ocge != _currentVisibilityExpression)
00493             {
00494                 if (_currentVisibilityExpression != null)
00495                 {
00496                     _figures.Add(new
      OptionalContentFigure(OptionalContentFigure.OptionalContentType.End, _currentVisibilityExpression));
00497                 }
00498
00499                 if (ocge != null)
00500                 {
00501                     _figures.Add(new
      OptionalContentFigure(OptionalContentFigure.OptionalContentType.Start, ocge));
00502                     this.VisibilityExpressions[ocge.ToString()] = ocge;
00503                 }
00504
00505                 _currentVisibilityExpression = ocge;
00506             }
00507
00508             _figures.Add(new TransformFigure(TransformFigure.TransformTypes.Save, null, this.Tag));
00509         }
00510
00511
00512         public void StrokeText(string text, double x, double y)
00513         {
00514             if (!_textToPaths)
00515             {
00516                 OptionalContentGroupExpression ocge = null;
00517
00518                 if (_ocgVisibilityExpressions != null && this.Tag != null)
00519                 {
00520                     _ocgVisibilityExpressions.TryGetValue(this.Tag, out ocge);
00521                 }
00522
00523                 if (ocge != _currentVisibilityExpression)
00524                 {
00525                     if (_currentVisibilityExpression != null)
00526                     {
00527                         _figures.Add(new
      OptionalContentFigure(OptionalContentFigure.OptionalContentType.End, _currentVisibilityExpression));
00528                     }
00529
00530                     if (ocge != null)
00531                     {
00532                         _figures.Add(new
```

```
OptionalContentFigure(OptionalContentFigure.OptionalContentType.Start, ocge));
00533                         this.VisibilityExpressions[ocge.ToString()] = ocge;
00534                     }
00535
00536                     _currentVisibilityExpression = ocge;
00537                 }
00538
00539                 if (!FontFamilies.TryGetValue(this.Font.FontFamily.FileName, out (FontFamily,
HashSet<char>) usedChars))
00540                 {
00541                     usedChars = (this.Font.FontFamily, new HashSet<char>());
00542                     FontFamilies[this.Font.FontFamily.FileName] = usedChars;
00543                 }
00544
00545                 foreach (char c in text)
00546                 {
00547                     usedChars.Item2.Add(c);
00548                 }
00549
00550                 _figures.Add(new TextFigure(text, Font, new Point(x, y), TextBaseline, null,
_strokeStyle, LineWidth, LineCap, LineJoin, _lineDash, this.Tag));
00551             }
00552             else
00553             {
00554                 PathText(text, x, y);
00555                 Stroke();
00556             }
00557
00558             if (this.StrokeStyle is SolidColourBrush solidColourBrush)
00559             {
00560                 this.Alphas.Add(solidColourBrush.A);
00561             }
00562             else if (this.StrokeStyle is GradientBrush gradientBrush)
00563             {
00564                 foreach (GradientStop stop in gradientBrush.GradientStops)
00565                 {
00566                     this.Alphas.Add(stop.Colour.A);
00567                 }
00568             }
00569         }
00570
00571         public void Translate(double x, double y)
00572         {
00573             OptionalContentGroupExpression ocge = null;
00574
00575             if (_ocgVisibilityExpressions != null && this.Tag != null)
00576             {
00577                 _ocgVisibilityExpressions.TryGetValue(this.Tag, out ocge);
00578             }
00579
00580             if (ocge != _currentVisibilityExpression)
00581             {
00582                 if (_currentVisibilityExpression != null)
00583                 {
00584                     _figures.Add(new
OptionalContentFigure(OptionalContentFigure.OptionalContentType.End, _currentVisibilityExpression));
00585                 }
00586
00587                 if (ocge != null)
00588                 {
00589                     _figures.Add(new
OptionalContentFigure(OptionalContentFigure.OptionalContentType.Start, ocge));
00590                     this.VisibilityExpressions[ocge.ToString()] = ocge;
00591                 }
00592
00593                 _currentVisibilityExpression = ocge;
00594             }
00595             _figures.Add(new TransformFigure(TransformFigure.TransformTypes.Transform, new double[,] {
{ 1, 0, x }, { 0, 1, y }, { 0, 0, 1 } }, this.Tag));
00596         }
00597
00598         public void Scale(double scaleX, double scaleY)
00599         {
00600             OptionalContentGroupExpression ocge = null;
00601
00602             if (_ocgVisibilityExpressions != null && this.Tag != null)
00603             {
00604                 _ocgVisibilityExpressions.TryGetValue(this.Tag, out ocge);
00605             }
00606
00607             if (ocge != _currentVisibilityExpression)
00608             {
00609                 if (_currentVisibilityExpression != null)
00610                 {
00611                     _figures.Add(new
OptionalContentFigure(OptionalContentFigure.OptionalContentType.End, _currentVisibilityExpression));
00612                 }
```

```
00613
00614                 if (ocge != null)
00615                 {
00616                     _figures.Add(new
      OptionalContentFigure(OptionalContentFigure.OptionalContentType.Start, ocge));
00617                     this.VisibilityExpressions[ocge.ToString()] = ocge;
00618                 }
00619
00620                 _currentVisibilityExpression = ocge;
00621             }
00622             _figures.Add(new TransformFigure(TransformFigure.TransformTypes.Transform, new double[,] {
      { scaleX, 0, 0 }, { 0, scaleY, 0 }, { 0, 0, 1 } }, this.Tag));
00623         }
00624
00625         public void Transform(double a, double b, double c, double d, double e, double f)
00626         {
00627             OptionalContentGroupExpression ocge = null;
00628
00629             if (_ocgVisibilityExpressions != null && this.Tag != null)
00630             {
00631                 _ocgVisibilityExpressions.TryGetValue(this.Tag, out ocge);
00632             }
00633
00634             if (ocge != _currentVisibilityExpression)
00635             {
00636                 if (_currentVisibilityExpression != null)
00637                 {
00638                     _figures.Add(new
      OptionalContentFigure(OptionalContentFigure.OptionalContentType.End, _currentVisibilityExpression));
00639                 }
00640
00641                 if (ocge != null)
00642                 {
00643                     _figures.Add(new
      OptionalContentFigure(OptionalContentFigure.OptionalContentType.Start, ocge));
00644                     this.VisibilityExpressions[ocge.ToString()] = ocge;
00645                 }
00646
00647                 _currentVisibilityExpression = ocge;
00648             }
00649             _figures.Add(new TransformFigure(TransformFigure.TransformTypes.Transform, new double[,] {
      { a, b, e }, { c, d, f }, { 0, 0, 1 } }, this.Tag));
00650         }
00651
00652         public void DrawRasterImage(int sourceX, int sourceY, int sourceWidth, int sourceHeight,
      double destinationX, double destinationY, double destinationWidth, double destinationHeight,
      RasterImage image)
00653         {
00654             OptionalContentGroupExpression ocge = null;
00655
00656             if (_ocgVisibilityExpressions != null && this.Tag != null)
00657             {
00658                 _ocgVisibilityExpressions.TryGetValue(this.Tag, out ocge);
00659             }
00660
00661             if (ocge != _currentVisibilityExpression)
00662             {
00663                 if (_currentVisibilityExpression != null)
00664                 {
00665                     _figures.Add(new
      OptionalContentFigure(OptionalContentFigure.OptionalContentType.End, _currentVisibilityExpression));
00666                 }
00667
00668                 if (ocge != null)
00669                 {
00670                     _figures.Add(new
      OptionalContentFigure(OptionalContentFigure.OptionalContentType.Start, ocge));
00671                     this.VisibilityExpressions[ocge.ToString()] = ocge;
00672                 }
00673
00674                 _currentVisibilityExpression = ocge;
00675             }
00676
00677             Save();
00678
00679             MoveTo(destinationX, destinationY);
00680             LineTo(destinationX + destinationWidth, destinationY);
00681             LineTo(destinationX + destinationWidth, destinationY + destinationHeight);
00682             LineTo(destinationX, destinationY + destinationHeight);
00683             Close();
00684             SetClippingPath();
00685
00686             double sourceRectX = (double)sourceX / image.Width;
00687             double sourceRectY = 1 - (double)sourceY / image.Height;
00688             double sourceRectWidth = (double)sourceWidth / image.Width;
00689             double sourceRectHeight = -(double)sourceHeight / image.Height;
00690
```

```
00691             double scaleX = destinationWidth / sourceRectWidth;
00692             double scaleY = destinationHeight / sourceRectHeight;
00693 
00694             double translationX = destinationX / scaleX - sourceRectX;
00695             double translationY = destinationY / scaleY - sourceRectY;
00696 
00697 
00698             Scale(scaleX, scaleY);
00699             Translate(translationX, translationY);
00700 
00701             _figures.Add(new RasterImageFigure(image, this.Tag));
00702 
00703             this.Images[image.Id] = image;
00704 
00705             Restore();
00706         }
00707 
00708         public void DrawFilteredGraphics(Graphics graphics, IFilter filter)
00709         {
00710             if (this._filterOption.Operation ==
      PDFContextInterpreter.FilterOption.FilterOperations.RasteriseAll)
00711             {
00712                 double scale = this._filterOption.RasterisationResolution;
00713 
00714                 Rectangle bounds = graphics.GetBounds();
00715 
00716                 bounds = new Rectangle(bounds.Location.X - filter.TopLeftMargin.X, bounds.Location.Y -
      filter.TopLeftMargin.Y, bounds.Size.Width + filter.TopLeftMargin.X + filter.BottomRightMargin.X,
      bounds.Size.Height + filter.TopLeftMargin.Y + filter.BottomRightMargin.Y);
00717 
00718                 if (bounds.Size.Width > 0 && bounds.Size.Height > 0)
00719                 {
00720                     if (!this._filterOption.RasterisationResolutionRelative)
00721                     {
00722                         scale /= Math.Min(bounds.Size.Width, bounds.Size.Height);
00723                     }
00724 
00725                     if (graphics.TryRasterise(bounds, scale, true, out RasterImage rasterised))
00726                     {
00727                         RasterImage filtered = null;
00728 
00729                         if (filter is ILocationInvariantFilter locInvFilter)
00730                         {
00731                             filtered = locInvFilter.Filter(rasterised, scale);
00732                         }
00733                         else if (filter is IFilterWithLocation filterWithLoc)
00734                         {
00735                             filtered = filterWithLoc.Filter(rasterised, bounds, scale);
00736                         }
00737 
00738                         if (filtered != null)
00739                         {
00740                             rasterised.Dispose();
00741 
00742                             DrawRasterImage(0, 0, filtered.Width, filtered.Height, bounds.Location.X,
      bounds.Location.Y, bounds.Size.Width, bounds.Size.Height, filtered);
00743                         }
00744                     }
00745                     else
00746                     {
00747                         throw new NotImplementedException(@"The filter could not be rasterised!  You
      can avoid this error by doing one of the following:
00748 • Add a reference to VectSharp.Raster or VectSharp.Raster.ImageSharp (you may also need to add a using
      directive somewhere to force the assembly to be loaded).
00749 • Provide your own implementation of Graphics.RasterisationMethod.
00750 • Set the FilterOption.Operation to ""IgnoreAll"" or ""SkipAll"".");
00751                     }
00752                 }
00753             }
00754             else if (this._filterOption.Operation ==
      PDFContextInterpreter.FilterOption.FilterOperations.IgnoreAll)
00755             {
00756                 graphics.CopyToIGraphicsContext(this);
00757             }
00758             else
00759             {
00760 
00761             }
00762         }
00763     }
00764 }
```

8.32 PDFContextInterpreter.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2024 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.IO;
00021 using System.Text;
00022 using VectSharp.PDF.PDFObjects;
00023 using System.Linq;
00024 using VectSharp.PDF.Figures;
00025 using VectSharp.PDF.OptionalContentGroups;
00026
00027 namespace VectSharp.PDF
00028 {
00029     internal class AssetTagManager
00030     {
00031         private readonly Random Random;
00032         private readonly HashSet<string> TagCache = new HashSet<string>();
00033         private int FontIndex = 1;
00034         private int PatternIndex = 1;
00035         private int ImageIndex = 1;
00036         private int OptionalContentGroupMembershipIndex = 1;
00037
00038         public AssetTagManager(Random random = null)
00039         {
00040             this.Random = random ??  new Random();
00041         }
00042
00043         public string GetFontNameTag()
00044         {
00045             string randString = new string(Enumerable.Range(0, 6).Select(x => (char)Random.Next(65,
      91)).ToArray());
00046             while (TagCache.Contains(randString))
00047             {
00048                 randString = new string(Enumerable.Range(0, 6).Select(x => (char)Random.Next(65,
      91)).ToArray());
00049             }
00050
00051             return randString;
00052         }
00053
00054         public string GetFontReferenceName()
00055         {
00056             string fontId = "F" + FontIndex.ToString();
00057             FontIndex++;
00058             return fontId;
00059         }
00060
00061         public string GetGradientAlphaReferenceName()
00062         {
00063             string gradientAlphaId = "pa" + PatternIndex.ToString();
00064             PatternIndex++;
00065             return gradientAlphaId;
00066         }
00067
00068         public string GetImageReferenceName()
00069         {
00070             string imageId = "Img" + ImageIndex.ToString();
00071             ImageIndex++;
00072             return imageId;
00073         }
00074
00075         public string GetOptionalGroupMembershipReferenceName()
00076         {
00077             string ocId = "oc" + OptionalContentGroupMembershipIndex.ToString();
00078             OptionalContentGroupMembershipIndex++;
00079             return ocId;
00080         }
00081     }
00082
00083 /// <summary>
```

```
00084 /// Contains methods to render a <see cref="Document"/> as a PDF document.
00085 /// </summary>
00086     public static class PDFContextInterpreter
00087     {
00088 /// <summary>
00089 /// Defines whether the used fonts should be included in the file.
00090 /// </summary>
00091         public enum TextOptions
00092         {
00093 /// <summary>
00094 /// Embeds subsetted font files containing only the glyphs for the characters that have been used.
00095 /// </summary>
00096             SubsetFonts,
00097
00098 /// <summary>
00099 /// Does not embed any font file and converts all text items into paths.
00100 /// </summary>
00101             ConvertIntoPaths
00102         }
00103
00104 /// <summary>
00105 /// Determines how and whether image filters are rasterised.
00106 /// </summary>
00107         public class FilterOption
00108         {
00109 /// <summary>
00110 /// Defines whether image filters should be rasterised or not.
00111 /// </summary>
00112             public enum FilterOperations
00113             {
00114 /// <summary>
00115 /// Image filters will always be rasterised.
00116 /// </summary>
00117                 RasteriseAll,
00118
00119 /// <summary>
00120 /// All image filters will be ignored.
00121 /// </summary>
00122                 IgnoreAll,
00123
00124 /// <summary>
00125 /// All the images that should be drawn with a filter will be ignored.
00126 /// </summary>
00127                 SkipAll
00128             }
00129
00130 /// <summary>
00131 /// Defines whether image filters should be rasterised or not.
00132 /// </summary>
00133             public FilterOperations Operation { get; } = FilterOperations.RasteriseAll;
00134
00135 /// <summary>
00136 /// The resolution that will be used to rasterise image filters.  Depending on the value of <see
      cref="RasterisationResolutionRelative"/>, this can either be an absolute resolution (i.e.  a size in
      pixel), or a scale factor that is applied to the image size in graphics units.
00137 /// </summary>
00138             public double RasterisationResolution { get; } = 1;
00139
00140 /// <summary>
00141 /// Determines whether the value of <see cref="RasterisationResolution"/> is absolute (i.e.  a size in
      pixel), or relative (i.e.  a scale factor that is applied to the image size in graphics units).
00142 /// </summary>
00143             public bool RasterisationResolutionRelative { get; } = true;
00144
00145 /// <summary>
00146 /// The default options for image filter rasterisation.
00147 /// </summary>
00148             public static FilterOption Default = new FilterOption(FilterOperations.RasteriseAll, 1,
      true);
00149
00150 /// <summary>
00151 /// Create a new <see cref="FilterOption"/> object.
00152 /// </summary>
00153 /// <param name="operation">Defines whether image filters should be rasterised or not.</param>
00154 /// <param name="rasterisationResolution">The resolution that will be used to rasterise image filters.
      Depending on the value of <see cref="RasterisationResolutionRelative"/>, this can either be an
      absolute resolution (i.e.  a size in pixel), or a scale factor that is applied to the image size in
      graphics units.</param>
00155 /// <param name="rasterisationResolutionRelative">Determines whether the value of <see
      cref="RasterisationResolution"/> is absolute (i.e.  a size in pixel), or relative (i.e.  a scale
      factor that is applied to the image size in graphics units).</param>
00156             public FilterOption(FilterOperations operation, double rasterisationResolution, bool
      rasterisationResolutionRelative)
00157             {
00158                 this.Operation = operation;
00159                 this.RasterisationResolution = rasterisationResolution;
00160                 this.RasterisationResolutionRelative = rasterisationResolutionRelative;
```

```
00161             }
00162         }
00163
00164         internal static readonly char[] CP1252Chars = new char[] { '\u0000', '\u0001', '\u0002',
      '\u0003', '\u0004', '\u0005', '\u0006', '\u0007', '\u0008', '\u0009', '\u000A', '\u000B', '\u000C',
      '\u000D', '\u000E', '\u000F', '\u0010', '\u0011', '\u0012', '\u0013', '\u0014', '\u0015', '\u0016',
      '\u0017', '\u0018', '\u0019', '\u001A', '\u001B', '\u001C', '\u001D', '\u001E', '\u001F', '\u0020',
      '\u0021', '\u0022', '\u0023', '\u0024', '\u0025', '\u0026', '\u0027', '\u0028', '\u0029', '\u002A',
      '\u002B', '\u002C', '\u002D', '\u002E', '\u002F', '\u0030', '\u0031', '\u0032', '\u0033', '\u0034',
      '\u0035', '\u0036', '\u0037', '\u0038', '\u0039', '\u003A', '\u003B', '\u003C', '\u003D', '\u003E',
      '\u003F', '\u0040', '\u0041', '\u0042', '\u0043', '\u0044', '\u0045', '\u0046', '\u0047', '\u0048',
      '\u0049', '\u004A', '\u004B', '\u004C', '\u004D', '\u004E', '\u004F', '\u0050', '\u0051', '\u0052',
      '\u0053', '\u0054', '\u0055', '\u0056', '\u0057', '\u0058', '\u0059', '\u005A', '\u005B', '\u005C',
      '\u005D', '\u005E', '\u005F', '\u0060', '\u0061', '\u0062', '\u0063', '\u0064', '\u0065', '\u0066',
      '\u0067', '\u0068', '\u0069', '\u006A', '\u006B', '\u006C', '\u006D', '\u006E', '\u006F', '\u0070',
      '\u0071', '\u0072', '\u0073', '\u0074', '\u0075', '\u0076', '\u0077', '\u0078', '\u0079', '\u007A',
      '\u007B', '\u007C', '\u007D', '\u007E', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000',
      '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000',
      '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000',
      '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u00A0', '\u00A1', '\u00A2',
      '\u00A3', '\u0000', '\u00A5', '\u0000', '\u00A7', '\u0000', '\u0000', '\u0000', '\u00AB', '\u0000',
      '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u00B5', '\u00B6',
      '\u0000', '\u0000', '\u0000', '\u0000', '\u00BB', '\u0000', '\u0000', '\u0000', '\u00BE', '\u00BF',
      '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u0000', '\u00C9', '\u0000',
      '\u0000', '\u00CC', '\u0000', '\u0000', '\u0000', '\u0000', '\u00D1', '\u00D2', '\u00D3', '\u00D4',
      '\u00D5', '\u00D6', '\u00D7', '\u00D8', '\u00D9', '\u00DA', '\u00DB', '\u00DC', '\u00DD', '\u00DE',
      '\u00DF', '\u00E0', '\u0000', '\u00E2', '\u0000', '\u00E4', '\u00E5', '\u00E6', '\u00E7', '\u0000',
      '\u0000', '\u0000', '\u0000', '\u00EC', '\u00ED', '\u00EE', '\u00EF', '\u00F0', '\u0000', '\u00F2',
      '\u00F3', '\u00F4', '\u0000', '\u00F6', '\u00F7', '\u0000', '\u0000', '\u0000', '\u0000', '\u00FC',
      '\u00FD', '\u00FE', '\u00FF' };
00165
00166         private static Dictionary<(string fontFamilyName, bool isSymbolic), PDFFont>
      GenerateFontObjects(Dictionary<string, (FontFamily, HashSet<char>)> fontFamilies, AssetTagManager
      tagManager, List<PDFReferenceableObject> pdfObjects, bool compressStreams, out PDFRawDictionary
      fontList)
00167         {
00168             Dictionary<(string fontFamilyName, bool isSymbolic), PDFFont> fontObjects = new
      Dictionary<(string fontFamilyName, bool isSymbolic), PDFFont>();
00169
00170             if (fontFamilies.Count > 0)
00171             {
00172                 fontList = new PDFRawDictionary();
00173             }
00174             else
00175             {
00176                 fontList = null;
00177             }
00178
00179             foreach (KeyValuePair<string, (FontFamily, HashSet<char>)> fontFamily in fontFamilies)
00180             {
00181                 List<char> nonSymbol = new List<char>();
00182                 List<char> symbol = new List<char>();
00183
00184                 foreach (char c in fontFamily.Value.Item2)
00185                 {
00186                     if (CP1252Chars.Contains(c))
00187                     {
00188                         nonSymbol.Add(c);
00189                     }
00190                     else
00191                     {
00192                         symbol.Add(c);
00193                     }
00194                 }
00195
00196                 if ((fontFamily.Value.Item1.IsStandardFamily && fontFamily.Value.Item1.FileName !=
      "Symbol" && fontFamily.Value.Item1.FileName != "ZapfDingbats" && symbol.Count == 0) ||
      fontFamily.Value.Item1.TrueTypeFile == null)
00197                 {
00198                     PDFType1Font font = new PDFType1Font(fontFamily.Value.Item1.IsStandardFamily ?
      fontFamily.Value.Item1.FileName.Replace(" ", "-") :  fontFamily.Value.Item1.FamilyName,
      tagManager.GetFontReferenceName());
00199                     pdfObjects.Add(font);
00200                     fontObjects[(fontFamily.Value.Item1.FileName, true)] = font;
00201                     fontObjects[(fontFamily.Value.Item1.FileName, false)] = font;
00202                     fontList.Keys[font.FontReferenceName] = font;
00203                 }
00204                 else
00205                 {
00206                     TrueTypeFile subsettedFont = fontFamily.Value.Item1.TrueTypeFile.SubsetFont(new
      string(fontFamily.Value.Item2.ToArray()));
00207
00208                     PDFTTFStream ttfStream = new PDFTTFStream(subsettedFont, compressStreams);
00209                     pdfObjects.Add(ttfStream);
00210
00211                     if (nonSymbol.Count > 0)
00212                     {
```

```
00213                         if (fontFamily.Value.Item1.IsStandardFamily && fontFamily.Value.Item1.FileName
      != "Symbol" && fontFamily.Value.Item1.FileName != "ZapfDingbats")
00214                         {
00215                             PDFType1Font font = new
      PDFType1Font(fontFamily.Value.Item1.IsStandardFamily ?  fontFamily.Value.Item1.FileName.Replace(" ",
      "-") :  fontFamily.Value.Item1.FamilyName, tagManager.GetFontReferenceName());
00216                             pdfObjects.Add(font);
00217                             fontObjects[(fontFamily.Value.Item1.FileName, true)] = font;
00218                             fontObjects[(fontFamily.Value.Item1.FileName, false)] = font;
00219                             fontList.Keys[font.FontReferenceName] = font;
00220                         }
00221                         else
00222                         {
00223                             PDFFontDescriptor fontDescriptor = new PDFFontDescriptor(subsettedFont,
      false, ttfStream, tagManager.GetFontNameTag());
00224                             pdfObjects.Add(fontDescriptor);
00225 
00226                             PDFTrueTypeFont font = new PDFTrueTypeFont(nonSymbol, fontDescriptor,
      subsettedFont, tagManager.GetFontReferenceName());
00227                             pdfObjects.Add(font);
00228                             fontObjects[(fontFamily.Value.Item1.FileName, false)] = font;
00229                             fontList.Keys[font.FontReferenceName] = font;
00230                         }
00231                     }
00232 
00233 
00234                     if (symbol.Count > 0)
00235                     {
00236                         PDFFontDescriptor fontDescriptor = new PDFFontDescriptor(subsettedFont, true,
      ttfStream, tagManager.GetFontNameTag());
00237                         pdfObjects.Add(fontDescriptor);
00238 
00239                         PDFCIDFontType2Font cidFont = new PDFCIDFontType2Font(symbol, fontDescriptor,
      subsettedFont, null);
00240                         pdfObjects.Add(cidFont);
00241 
00242                         PDFToUnicodeStream toUnicodeStream = new PDFToUnicodeStream(symbol,
      cidFont.GlyphIndices, compressStreams);
00243                         pdfObjects.Add(toUnicodeStream);
00244 
00245                         PDFType0Font font = new PDFType0Font(fontDescriptor, cidFont, toUnicodeStream,
      tagManager.GetFontReferenceName());
00246                         pdfObjects.Add(font);
00247                         fontObjects[(fontFamily.Value.Item1.FileName, true)] = font;
00248                         fontList.Keys[font.FontReferenceName] = font;
00249                     }
00250 
00251                     subsettedFont.Destroy();
00252                 }
00253             }
00254 
00255             if (fontList != null)
00256             {
00257                 pdfObjects.Add(fontList);
00258             }
00259 
00260             return fontObjects;
00261         }
00262 
00263         private static Dictionary<string, PDFImage> GenerateImageObjects(Dictionary<string,
      RasterImage> allImages, AssetTagManager tagManager, List<PDFReferenceableObject> pdfObjects, bool
      compressStreams)
00264         {
00265             Dictionary<string, PDFImage> imageObjects = new Dictionary<string, PDFImage>();
00266 
00267             foreach (KeyValuePair<string, RasterImage> kvp in allImages)
00268             {
00269                 PDFImageAlpha imageAlpha = null;
00270 
00271                 if (kvp.Value.HasAlpha)
00272                 {
00273                     imageAlpha = new PDFImageAlpha(kvp.Value, compressStreams);
00274                     pdfObjects.Add(imageAlpha);
00275                 }
00276 
00277                 PDFImage image = new PDFImage(kvp.Value, imageAlpha, compressStreams,
      tagManager.GetImageReferenceName());
00278                 pdfObjects.Add(image);
00279 
00280                 imageObjects[kvp.Key] = image;
00281             }
00282 
00283             return imageObjects;
00284         }
00285 
00286         private static PDFStream[] GeneratePageContentStreams(PDFContext[] pageContexts,
      Dictionary<(string fontFamilyName, bool isSymbolic), PDFFont> fontObjects, double[] alphas,
```

```
      Dictionary<string, PDFImage> imageObjects, bool compressStreams, List<PDFReferenceableObject>
      pdfObjects, out Dictionary<string, List<Rectangle»[] taggedObjectRectsByPage, out List<(GradientBrush,
      double[,], IFigure)> gradients, Dictionary<string, PDFOptionalContentGroupMembership> contentGroups)
00287         {
00288             PDFStream[] pageContentStreams = new PDFStream[pageContexts.Length];
00289 
00290             taggedObjectRectsByPage = new Dictionary<string, List<Rectangle»[pageContexts.Length];
00291             gradients = new List<(GradientBrush, double[,], IFigure)>();
00292 
00293             for (int pageInd = 0; pageInd < pageContexts.Length; pageInd++)
00294             {
00295                 taggedObjectRectsByPage[pageInd] = new Dictionary<string, List<Rectangle»();
00296 
00297                 double[,] transformationMatrix = new double[3, 3] { { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0,
      1 } };
00298                 Stack<double[,]> savedStates = new Stack<double[,]>();
00299 
00300                 MemoryStream contentStream = new MemoryStream();
00301 
00302                 using (StreamWriter ctW = new StreamWriter(contentStream, Encoding.ASCII, 1024, true))
00303                 {
00304                     for (int i = 0; i < pageContexts[pageInd]._figures.Count; i++)
00305                     {
00306                         bool isTransform = pageContexts[pageInd]._figures[i] is TransformFigure;
00307 
00308                         bool isClip = pageContexts[pageInd]._figures[i] is PathFigure pathFig &&
      pathFig.IsClipping;
00309 
00310                         if (!string.IsNullOrEmpty(pageContexts[pageInd]._figures[i].Tag) &&
      !isTransform && !isClip)
00311                         {
00312                             Rectangle boundingRect =
      PDFProcessFigure.MeasureFigure(pageContexts[pageInd]._figures[i], ref transformationMatrix,
      savedStates);
00313 
00314                             if
      (!taggedObjectRectsByPage[pageInd].TryGetValue(pageContexts[pageInd]._figures[i].Tag, out
      List<Rectangle> rects))
00315                             {
00316                                 rects = new List<Rectangle>();
00317 
      taggedObjectRectsByPage[pageInd].Add(pageContexts[pageInd]._figures[i].Tag, rects);
00318                             }
00319                             rects.Add(boundingRect);
00320                         }
00321                         else if (isTransform)
00322                         {
00323                             PDFProcessFigure.MeasureFigure(pageContexts[pageInd]._figures[i], ref
      transformationMatrix, savedStates);
00324                         }
00325 
00326                         PDFProcessFigure.WriteFigure(pageContexts[pageInd]._figures[i], fontObjects,
      alphas, imageObjects, transformationMatrix, gradients, contentGroups, ctW);
00327                     }
00328                 }
00329 
00330                 contentStream.Seek(0, SeekOrigin.Begin);
00331 
00332                 PDFStream pageStream = new PDFStream(contentStream, compressStreams);
00333                 pdfObjects.Add(pageStream);
00334 
00335                 pageContentStreams[pageInd] = pageStream;
00336             }
00337 
00338             return pageContentStreams;
00339         }
00340 
00341         private static PDFGradient CreateGradient(bool includeMatrix, GradientBrush gradient,
      double[,] matrix, List<PDFReferenceableObject> pdfObjects)
00342         {
00343             PDFInterpolationFunction functionObject;
00344 
00345             if (gradient.GradientStops.Count == 2)
00346             {
00347                 functionObject = new
      PDFExponentialInterpolationFunction(gradient.GradientStops[0].Colour,
      gradient.GradientStops[1].Colour);
00348                 pdfObjects.Add(functionObject);
00349             }
00350             else
00351             {
00352                 List<double> bounds = new List<double>();
00353                 List<PDFExponentialInterpolationFunction> functions = new
      List<PDFExponentialInterpolationFunction>();
00354 
00355                 for (int j = 0; j < gradient.GradientStops.Count - 1; j++)
00356                 {
```

```
00357                     PDFExponentialInterpolationFunction function2 = new
      PDFExponentialInterpolationFunction(gradient.GradientStops[j].Colour, gradient.GradientStops[j +
      1].Colour);
00358                     pdfObjects.Add(function2);
00359                     functions.Add(function2);
00360
00361                     if (j < gradient.GradientStops.Count - 2)
00362                     {
00363                         bounds.Add(gradient.GradientStops[j + 1].Offset);
00364                     }
00365                 }
00366
00367                 functionObject = new PDFStitchingFunction(functions, bounds);
00368                 pdfObjects.Add(functionObject);
00369             }
00370
00371             if (gradient is LinearGradientBrush linear)
00372             {
00373                 PDFLinearGradient pdfLin = new PDFLinearGradient(includeMatrix ?  matrix :  null,
      functionObject, linear);
00374                 pdfObjects.Add(pdfLin);
00375                 return pdfLin;
00376             }
00377             else if (gradient is RadialGradientBrush radial)
00378             {
00379                 PDFRadialGradient pdfRad = new PDFRadialGradient(includeMatrix ?  matrix :  null,
      functionObject, radial);
00380                 pdfObjects.Add(pdfRad);
00381                 return pdfRad;
00382             }
00383             else
00384             {
00385                 throw new NotImplementedException("Unknown type of gradient!");
00386             }
00387         }
00388
00389         private static List<(PDFGradient gradient, PDFGradientAlphaMask alphaMask)>
      GenerateGradients(List<(GradientBrush, double[,], IFigure)> gradients, AssetTagManager tagManager,
      List<PDFReferenceableObject> pdfObjects, bool compressStreams)
00390         {
00391             List<(PDFGradient gradient, PDFGradientAlphaMask alphaMask)> pdfGradients = new
      List<(PDFGradient gradient, PDFGradientAlphaMask alphaMask)>();
00392
00393             for (int i = 0; i < gradients.Count; i++)
00394             {
00395                 (GradientBrush gradient, double[,] matrix, IFigure figure) = gradients[i];
00396
00397                 PDFGradient pdfGradient = CreateGradient(true, gradient, matrix, pdfObjects);
00398
00399                 bool hasAlpha = gradient.GradientStops.Any(x => x.Colour.A != 1);
00400
00401                 if (hasAlpha)
00402                 {
00403                     GradientBrush alphaGradient = (GradientBrush)PDFContext.GetAlphaBrush(gradient);
00404
00405                     PDFGradient alphaPDFGradient = CreateGradient(false, alphaGradient, matrix,
      pdfObjects);
00406
00407                     string alphaPatternReferenceName = tagManager.GetGradientAlphaReferenceName();
00408
00409                     Rectangle bbox = figure.GetBounds();
00410
00411                     PDFGradientAlphaMaskXObject gradientMaskForm = new
      PDFGradientAlphaMaskXObject(bbox, alphaPatternReferenceName, alphaPDFGradient, compressStreams);
00412                     pdfObjects.Add(gradientMaskForm);
00413
00414                     PDFGradientAlphaMask gradientMask = new PDFGradientAlphaMask(gradientMaskForm);
00415                     pdfObjects.Add(gradientMask);
00416
00417                     pdfGradients.Add((pdfGradient, gradientMask));
00418                 }
00419                 else
00420                 {
00421                     pdfGradients.Add((pdfGradient, null));
00422                 }
00423             }
00424
00425             return pdfGradients;
00426         }
00427
00428
00429         private static (int page, Rectangle rect) GetTagRectangle(string tag, Dictionary<string,
      List<Rectangle»[] taggedObjectRectsByPage)
00430         {
00431             if (!string.IsNullOrEmpty(tag))
00432             {
00433                 for (int k = 0; k < taggedObjectRectsByPage.Length; k++)
```

```
00434                 {
00435                     if (taggedObjectRectsByPage[k].TryGetValue(tag, out List<Rectangle> target))
00436                     {
00437                         return (k, target[0]);
00438                     }
00439                 }
00440             }
00441 
00442             return (-1, new Rectangle());
00443         }
00444 
00445 
00446         private static void GenerateLinkAnnotations(PDFPage[] pages, Dictionary<string,
      List<Rectangle»[] taggedObjectRectsByPage, Dictionary<string, string> linkDestinations,
      Dictionary<string, PDFOptionalContentGroup> optionalContentGroups, Dictionary<string,
      OptionalContentGroupExpression> optionalContenGroupExpressions, Dictionary<string,
      PDFOptionalContentGroupMembership> optionalContentGroupMemberships, List<PDFReferenceableObject>
      pdfObjects, AnnotationStyleCollection annotationStyles)
00447         {
00448             for (int i = 0; i < pages.Length; i++)
00449             {
00450                 foreach (KeyValuePair<string, List<Rectangle» kvp in taggedObjectRectsByPage[i])
00451                 {
00452                     if (linkDestinations.TryGetValue(kvp.Key, out string destination))
00453                     {
00454                         if (destination.StartsWith("#"))
00455                         {
00456                             if (destination.StartsWith("#@OCG:"))
00457                             {
00458                                 destination = destination.Substring(6);
00459 
00460                                 string[] splitDestination = destination.Split(';');
00461 
00462                                 List<PDFOptionalContentGroup> on = new
      List<PDFOptionalContentGroup>();
00463                                 List<PDFOptionalContentGroup> off = new
      List<PDFOptionalContentGroup>();
00464                                 List<PDFOptionalContentGroup> toggle = new
      List<PDFOptionalContentGroup>();
00465 
00466                                 for (int j = 0; j < splitDestination.Length; j++)
00467                                 {
00468                                     string[] splitSplitDestination = splitDestination[j].Split(',');
00469 
00470                                     switch (splitSplitDestination[1].ToLowerInvariant())
00471                                     {
00472                                         case "on":
00473                                             on.Add(optionalContentGroups[splitSplitDestination[0]]);
00474                                             break;
00475                                         case "off":
00476                                             off.Add(optionalContentGroups[splitSplitDestination[0]]);
00477                                             break;
00478                                         case "toggle":
00479 
      toggle.Add(optionalContentGroups[splitSplitDestination[0]]);
00480                                             break;
00481                                     }
00482                                 }
00483 
00484                                 if (!annotationStyles.TryGetValue(kvp.Key, out AnnotationStyle style))
00485                                 {
00486                                     style = annotationStyles.DefaultStyle;
00487                                 }
00488 
00489                                 for (int l = 0; l < kvp.Value.Count; l++)
00490                                 {
00491                                     PDFSetOCGStateActionAnnotation annot = new
      PDFSetOCGStateActionAnnotation(kvp.Value[l], new PDFSetOCGStateAction(on, off, toggle),
      style.BorderWidth, style.BorderDash, style.BorderColour);
00492 
00493                                     if (optionalContenGroupExpressions.TryGetValue(kvp.Key, out
      OptionalContentGroupExpression ocge))
00494                                     {
00495                                         annot.OC = optionalContentGroupMemberships[ocge.ToString()];
00496                                     }
00497 
00498                                     pdfObjects.Add(annot);
00499                                     pages[i].AddAnnotation(annot);
00500                                 }
00501                             }
00502                             else
00503                             {
00504                                 (int pageNum, Rectangle target) =
      GetTagRectangle(destination.Substring(1), taggedObjectRectsByPage);
00505 
00506                                 if (pageNum >= 0)
00507                                 {
```

```
00508                                   if (!annotationStyles.TryGetValue(kvp.Key, out AnnotationStyle
      style))
00509                                   {
00510                                       style = annotationStyles.DefaultStyle;
00511                                   }
00512
00513                                   for (int l = 0; l < kvp.Value.Count; l++)
00514                                   {
00515                                       PDFInternalLinkAnnotation annot = new
      PDFInternalLinkAnnotation(kvp.Value[l], pages[pageNum], target.Location.X, target.Location.Y +
      target.Size.Height, style.BorderWidth, style.BorderDash, style.BorderColour);
00516
00517                                       if (optionalContentGroupMemberships.TryGetValue(kvp.Key, out
      PDFOptionalContentGroupMembership ocgm))
00518                                       {
00519                                           annot.OC = ocgm;
00520                                       }
00521
00522                                       pdfObjects.Add(annot);
00523                                       pages[i].AddAnnotation(annot);
00524                                   }
00525                               }
00526                           }
00527                       }
00528                       else
00529                       {
00530                           if (!annotationStyles.TryGetValue(kvp.Key, out AnnotationStyle style))
00531                           {
00532                               style = annotationStyles.DefaultStyle;
00533                           }
00534
00535                           for (int l = 0; l < kvp.Value.Count; l++)
00536                           {
00537                               PDFExternalLinkAnnotation annot = new
      PDFExternalLinkAnnotation(kvp.Value[l], destination, style.BorderWidth, style.BorderDash,
      style.BorderColour);
00538
00539                               if (optionalContentGroupMemberships.TryGetValue(kvp.Key, out
      PDFOptionalContentGroupMembership ocgm))
00540                               {
00541                                   annot.OC = ocgm;
00542                               }
00543
00544                               pdfObjects.Add(annot);
00545                               pages[i].AddAnnotation(annot);
00546                           }
00547                       }
00548                   }
00549               }
00550           }
00551       }
00552
00553       private static PDFOutlineItem CreateOutlineItem(OutlineTreeNode outlineNode, PDFPage[] pages,
      Dictionary<string, List<Rectangle>[] taggedObjectRectsByPage, List<PDFReferenceableObject> pdfObjects)
00554       {
00555           (int pageNum, Rectangle target) = GetTagRectangle(outlineNode.DestinationTag,
      taggedObjectRectsByPage);
00556
00557           PDFOutlineItem item;
00558
00559           if (pageNum >= 0)
00560           {
00561               item = new PDFOutlineItem(outlineNode.Title, pages[pageNum], target.Location.X,
      target.Location.Y + target.Size.Height, colour:  outlineNode.Colour, bold:  outlineNode.Bold, italic:
      outlineNode.Italic);
00562           }
00563           else
00564           {
00565               item = new PDFOutlineItem(outlineNode.Title, colour:  outlineNode.Colour, bold:
      outlineNode.Bold, italic:  outlineNode.Italic);
00566           }
00567
00568           if (outlineNode.Children.Count > 0)
00569           {
00570               PDFOutlineItem[] children = new PDFOutlineItem[outlineNode.Children.Count];
00571
00572               for (int i = 0; i < outlineNode.Children.Count; i++)
00573               {
00574                   children[i] = CreateOutlineItem(outlineNode.Children[i], pages,
      taggedObjectRectsByPage, pdfObjects);
00575               }
00576
00577               for (int i = 0; i < children.Length; i++)
00578               {
00579                   if (i > 0)
00580                   {
00581                       children[i].Prev = children[i - 1];
```

```
00582                     }
00583
00584                     if (i < children.Length - 1)
00585                     {
00586                         children[i].Next = children[i + 1];
00587                     }
00588
00589                     children[i].Parent = item;
00590                 }
00591
00592                 item.First = children[0];
00593                 item.Last = children[children.Length - 1];
00594             }
00595
00596             pdfObjects.Add(item);
00597             return item;
00598         }
00599
00600         private static PDFOutline GenerateOutline(OutlineTree outline, PDFPage[] pages,
      Dictionary<string, List<Rectangle»[] taggedObjectRectsByPage, List<PDFReferenceableObject> pdfObjects)
00601         {
00602             PDFOutlineItem[] topLevelItems = new PDFOutlineItem[outline.TopLevelItems.Count];
00603
00604             for (int i = 0; i < outline.TopLevelItems.Count; i++)
00605             {
00606                 topLevelItems[i] = CreateOutlineItem(outline.TopLevelItems[i], pages,
      taggedObjectRectsByPage, pdfObjects);
00607             }
00608
00609             PDFOutline pdfOutline = new PDFOutline(topLevelItems[0],
      topLevelItems[topLevelItems.Length - 1]);
00610             pdfObjects.Add(pdfOutline);
00611
00612             for (int i = 0; i < topLevelItems.Length; i++)
00613             {
00614                 if (i > 0)
00615                 {
00616                     topLevelItems[i].Prev = topLevelItems[i - 1];
00617                 }
00618
00619                 if (i < topLevelItems.Length - 1)
00620                 {
00621                     topLevelItems[i].Next = topLevelItems[i + 1];
00622                 }
00623
00624                 topLevelItems[i].Parent = pdfOutline;
00625             }
00626
00627             return pdfOutline;
00628         }
00629
00630         private static void CreateOptionalContentGroups(OptionalContentGroupExpression expression,
      Dictionary<string, PDFOptionalContentGroup> pdfOcgs, List<PDFReferenceableObject> pdfObjects,
      Dictionary<string, OptionalContentGroup> ocgs)
00631         {
00632             if (expression is OptionalContentGroup group)
00633             {
00634                 if (!pdfOcgs.ContainsKey(group.ToString()))
00635                 {
00636                     ocgs[group.ToString()] = group;
00637                     PDFOptionalContentGroup ocg = new PDFOptionalContentGroup(group.Name,
      group.Intent, group.DefaultViewState, group.DefaultPrintState, group.DefaultExportState);
00638                     pdfOcgs[group.ToString()] = ocg;
00639                     pdfObjects.Add(ocg);
00640                 }
00641             }
00642             else
00643             {
00644                 foreach (OptionalContentGroupExpression expr in expression.Arguments)
00645                 {
00646                     CreateOptionalContentGroups(expr, pdfOcgs, pdfObjects, ocgs);
00647                 }
00648             }
00649         }
00650
00651         private static IEnumerable<IPDFObject> CreateOCGOrder(OptionalContentGroupTreeNode node,
      Dictionary<string, PDFOptionalContentGroup> pdfOcgs)
00652         {
00653             if (node.Children.Count == 0)
00654             {
00655                 if (node.Label.LabelType == OptionalContentGroupTreeLabel.Type.OptionalContentGroup)
00656                 {
00657                     yield return pdfOcgs[node.Label.LabelOptionalContentGroup.ToString()];
00658                 }
00659                 else
00660                 {
00661                     throw new ArgumentException("A leaf node must have an optional content group label
```

```
and not a string label!");
00662                 }
00663             }
00664             else
00665             {
00666                 if (node.Label.LabelType == OptionalContentGroupTreeLabel.Type.String)
00667                 {
00668                     yield return new PDFArray<IPDFObject>(new IPDFObject[] { new
PDFString(node.Label.LabelString, PDFString.StringDelimiter.Brackets)
}.Concat(node.Children.SelectMany(x => CreateOCGOrder(x, pdfOcgs))));
00669                 }
00670                 else if (node.Label.LabelType ==
OptionalContentGroupTreeLabel.Type.OptionalContentGroup)
00671                 {
00672                     yield return pdfOcgs[node.Label.LabelOptionalContentGroup.ToString()];
00673                     yield return new PDFArray<IPDFObject>(node.Children.SelectMany(x =>
CreateOCGOrder(x, pdfOcgs)));
00674                 }
00675             }
00676         }
00677
00678         private static Dictionary<string, PDFOptionalContentGroupMembership>
GenerateOptionalContentGroups(Dictionary<string, OptionalContentGroupExpression> expressions,
OptionalContentGroupSettings groupSettings, IEnumerable<IEnumerable<OptionalContentGroup»
radioButtonGroups, AssetTagManager tagManager, List<PDFReferenceableObject> pdfObjects, out
Dictionary<string, PDFOptionalContentGroup> contentGroups, out PDFOptionalContentProperties
optionalContentProperties)
00679         {
00680             Dictionary<string, PDFOptionalContentGroupMembership> tbr = new Dictionary<string,
PDFOptionalContentGroupMembership>();
00681             Dictionary<string, PDFOptionalContentGroup> pdfOcgs = new Dictionary<string,
PDFOptionalContentGroup>();
00682             Dictionary<string, OptionalContentGroup> ocgs = new Dictionary<string,
OptionalContentGroup>();
00683
00684             foreach (KeyValuePair<string, OptionalContentGroupExpression> kvp in expressions)
00685             {
00686                 CreateOptionalContentGroups(kvp.Value, pdfOcgs, pdfObjects, ocgs);
00687
00688                 if (!tbr.ContainsKey(kvp.Key))
00689                 {
00690                     PDFOptionalContentGroupMembership ocgm = new
PDFOptionalContentGroupMembership(kvp.Value, pdfOcgs,
tagManager.GetOptionalGroupMembershipReferenceName());
00691                     tbr[kvp.Key] = ocgm;
00692                     pdfObjects.Add(ocgm);
00693                 }
00694             }
00695
00696             List<PDFUsageApplication> uas = new List<PDFUsageApplication>();
00697
00698             PDFUsageApplication uaView = new
PDFUsageApplication(PDFUsageApplication.UsageApplicationEvent.View, pdfOcgs.Values);
00699             if (uaView.OCGs.Values.Count > 0)
00700             {
00701                 pdfObjects.Add(uaView);
00702                 uas.Add(uaView);
00703             }
00704
00705             PDFUsageApplication uaPrint = new
PDFUsageApplication(PDFUsageApplication.UsageApplicationEvent.Print, pdfOcgs.Values);
00706             if (uaPrint.OCGs.Values.Count > 0)
00707             {
00708                 pdfObjects.Add(uaPrint);
00709                 uas.Add(uaPrint);
00710             }
00711
00712             PDFUsageApplication uaExport = new
PDFUsageApplication(PDFUsageApplication.UsageApplicationEvent.Export, pdfOcgs.Values);
00713             if (uaExport.OCGs.Values.Count > 0)
00714             {
00715                 pdfObjects.Add(uaExport);
00716                 uas.Add(uaExport);
00717             }
00718
00719             IEnumerable<IPDFObject> order = pdfOcgs.Values;
00720
00721             if (groupSettings?.OptionalContentGroupTree != null)
00722             {
00723                 order = groupSettings.OptionalContentGroupTree.SelectMany(x => CreateOCGOrder(x,
pdfOcgs));
00724             }
00725
00726             PDFOptionalContentConfiguration defaultConfiguration = new
PDFOptionalContentConfiguration(ocgs.Values.Where(x => !x.DefaultState).Select(x =>
pdfOcgs[x.ToString()]), uas, order, radioButtonGroups?.Select(x => x.Select(y =>
pdfOcgs[y.ToString()])), "Default");
```

```
00727             pdfObjects.Add(defaultConfiguration);
00728
00729             optionalContentProperties = new PDFOptionalContentProperties(pdfOcgs.Values,
      defaultConfiguration);
00730             contentGroups = pdfOcgs;
00731
00732             return tbr;
00733
00734         }
00735
00736 /// <summary>
00737 /// Convert the document to a <see cref="PDFDocument"/> representation.
00738 /// </summary>
00739 /// <param name="document">The <see cref="Document"/> to convert.</param>
00740 /// <param name="textOption">Defines whether the used fonts should be included in the PDF
      document.</param>
00741 /// <param name="compressStreams">Indicates whether the streams in the PDF document should be
      compressed.</param>
00742 /// <param name="linkDestinations">A dictionary associating element tags to link targets.  If this is
      provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to
      the destination specified in the dictionary.  If the destination starts with a hash (#), it is
      interpreted as the tag of another object in the current document; otherwise, it is interpreted as an
      external URI.</param>
00743 /// <param name="outline">Document outline (table of contents).</param>
00744 /// <param name="metadata">Document metadata.  Use <see cref="PDFMetadata.NoMetadata()"/> if you do
      not wish to include metadata in the document.</param>
00745 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
00746 /// <param name="optionalContentGroupSettings">Settings for optional content groups (layers).</param>
00747 /// <param name="annotationStyles">Annotation appearance styles.</param>
00748         public static PDFDocument CreatePDFDocument(this Document document, TextOptions textOption =
      TextOptions.SubsetFonts, bool compressStreams = true, Dictionary<string, string> linkDestinations =
      null, OutlineTree outline = null, PDFMetadata metadata = default, FilterOption filterOption = default,
      OptionalContentGroupSettings optionalContentGroupSettings = default, AnnotationStyleCollection
      annotationStyles = default)
00749         {
00750             if (linkDestinations == null)
00751             {
00752                 linkDestinations = new Dictionary<string, string>();
00753             }
00754
00755             if (filterOption == null)
00756             {
00757                 filterOption = FilterOption.Default;
00758             }
00759
00760             if (metadata == null)
00761             {
00762                 metadata = new PDFMetadata();
00763             }
00764
00765             if (optionalContentGroupSettings == null)
00766             {
00767                 optionalContentGroupSettings = new OptionalContentGroupSettings();
00768             }
00769
00770             if (annotationStyles == null)
00771             {
00772                 annotationStyles = new AnnotationStyleCollection();
00773             }
00774
00775             PDFContext[] pageContexts = new PDFContext[document.Pages.Count];
00776
00777             Dictionary<string, (FontFamily, HashSet<char>)> fontFamilies = new Dictionary<string,
      (FontFamily, HashSet<char>)>();
00778             Dictionary<string, RasterImage> allImages = new Dictionary<string, RasterImage>();
00779             HashSet<double> allAlphas = new HashSet<double>() { 0, 1 };
00780             Dictionary<string, OptionalContentGroupExpression> allVisibilityExpressions = new
      Dictionary<string, OptionalContentGroupExpression>();
00781
00782             for (int i = 0; i < document.Pages.Count; i++)
00783             {
00784                 pageContexts[i] = new PDFContext(document.Pages[i].Width, document.Pages[i].Height,
      document.Pages[i].Background, fontFamilies, allImages, allAlphas, textOption ==
      TextOptions.ConvertIntoPaths, filterOption, optionalContentGroupSettings.Groups,
      allVisibilityExpressions);
00785                 document.Pages[i].Graphics.CopyToIGraphicsContext(pageContexts[i]);
00786                 pageContexts[i].Finish();
00787             }
00788
00789             double[] alphas = allAlphas.ToArray();
00790
00791             AssetTagManager tagManager = new AssetTagManager();
00792             List<PDFReferenceableObject> pdfObjects = new List<PDFReferenceableObject>();
00793
00794             Dictionary<(string fontFamilyName, bool isSymbolic), PDFFont> fontObjects =
      GenerateFontObjects(fontFamilies, tagManager, pdfObjects, compressStreams, out PDFRawDictionary
```

```
      fontList);
00795             Dictionary<string, PDFImage> imageObjects = GenerateImageObjects(allImages, tagManager,
      pdfObjects, compressStreams);
00796
00797             PDFOptionalContentProperties optionalContentProperties = null;
00798             Dictionary<string, PDFOptionalContentGroupMembership> contentGroupMemberships = new
      Dictionary<string, PDFOptionalContentGroupMembership>();
00799             Dictionary<string, PDFOptionalContentGroup> contentGroups = new Dictionary<string,
      PDFOptionalContentGroup>();
00800
00801             if (allVisibilityExpressions.Count > 0)
00802             {
00803                 contentGroupMemberships = GenerateOptionalContentGroups(allVisibilityExpressions,
      optionalContentGroupSettings, optionalContentGroupSettings?.RadioButtonGroups, tagManager, pdfObjects,
      out contentGroups, out optionalContentProperties);
00804             }
00805
00806             PDFStream[] pageContentStreams = GeneratePageContentStreams(pageContexts, fontObjects,
      alphas, imageObjects, compressStreams, pdfObjects, out Dictionary<string, List<Rectangle»[]
      taggedObjectRectsByPage, out List<(GradientBrush, double[,], IFigure)> gradients,
      contentGroupMemberships);
00807
00808             List<(PDFGradient, PDFGradientAlphaMask)> pdfGradients = GenerateGradients(gradients,
      tagManager, pdfObjects, compressStreams);
00809
00810             PDFResources resources = new PDFResources(fontList, alphas, pdfGradients,
      imageObjects.Values, contentGroupMemberships);
00811             pdfObjects.Add(resources);
00812
00813             PDFPage[] pages = new PDFPage[document.Pages.Count];
00814
00815             for (int i = 0; i < document.Pages.Count; i++)
00816             {
00817                 pages[i] = new PDFPage(document.Pages[i].Width, document.Pages[i].Height, resources,
      pageContentStreams[i]);
00818                 pdfObjects.Add(pages[i]);
00819             }
00820
00821             GenerateLinkAnnotations(pages, taggedObjectRectsByPage, linkDestinations, contentGroups,
      optionalContentGroupSettings.Groups, contentGroupMemberships, pdfObjects, annotationStyles);
00822
00823             PDFPages pdfPages = new PDFPages(pages);
00824             pdfObjects.Add(pdfPages);
00825
00826             for (int i = 0; i < pages.Length; i++)
00827             {
00828                 pages[i].Parent = pdfPages;
00829             }
00830
00831             PDFCatalog catalog = new PDFCatalog(pdfPages);
00832             pdfObjects.Add(catalog);
00833
00834             if (outline != null)
00835             {
00836                 PDFOutline pdfOutline = GenerateOutline(outline, pages, taggedObjectRectsByPage,
      pdfObjects);
00837                 catalog.Outlines = pdfOutline;
00838                 catalog.PageMode = new PDFString("UseOutlines",
      PDFString.StringDelimiter.StartingForwardSlash);
00839             }
00840
00841             PDFDocument doc = new PDFDocument(pdfObjects);
00842
00843             if (optionalContentProperties != null)
00844             {
00845                 catalog.OCProperties = optionalContentProperties;
00846                 doc.PDFVersion = "1.6";
00847             }
00848
00849             PDFDocumentInfo info = metadata.ToPDFDocumentInfo();
00850
00851             if (info != null)
00852             {
00853                 doc.Info = info;
00854
00855                 if (info.Title != null)
00856                 {
00857                     catalog.ViewerPreferences = new PDFRawDictionary();
00858                     catalog.ViewerPreferences.Keys["DisplayDocTitle"] = new PDFBool(true);
00859                 }
00860             }
00861
00862             return doc;
00863         }
00864
00865 /// <summary>
00866 /// Save the document to a PDF file.
```

```
00867 /// </summary>
00868 /// <param name="document">The <see cref="Document"/> to save.</param>
00869 /// <param name="fileName">The full path to the file to save.  If it exists, it will be
      overwritten.</param>
00870 /// <param name="textOption">Defines whether the used fonts should be included in the file.</param>
00871 /// <param name="compressStreams">Indicates whether the streams in the PDF file should be
      compressed.</param>
00872 /// <param name="linkDestinations">A dictionary associating element tags to link targets.  If this is
      provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to
      the destination specified in the dictionary.  If the destination starts with a hash (#), it is
      interpreted as the tag of another object in the current document; otherwise, it is interpreted as an
      external URI.</param>
00873 /// <param name="outline">Document outline (table of contents).</param>
00874 /// <param name="metadata">Document metadata.  Use <see cref="PDFMetadata.NoMetadata()"/> if you do
      not wish to include metadata in the document.</param>
00875 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
00876 /// <param name="optionalContentGroupSettings">Settings for optional content groups (layers).</param>
00877 /// <param name="annotationStyles">Annotation appearance styles.</param>
00878         public static void SaveAsPDF(this Document document, string fileName, TextOptions textOption =
      TextOptions.SubsetFonts, bool compressStreams = true, Dictionary<string, string> linkDestinations =
      null, OutlineTree outline = null, PDFMetadata metadata = default, FilterOption filterOption = default,
      OptionalContentGroupSettings optionalContentGroupSettings = default, AnnotationStyleCollection
      annotationStyles = default)
00879         {
00880             using (FileStream stream = new FileStream(fileName, FileMode.Create))
00881             {
00882                 document.SaveAsPDF(stream, textOption, compressStreams, linkDestinations, outline,
      metadata, filterOption, optionalContentGroupSettings, annotationStyles);
00883             }
00884         }
00885 
00886 /// <summary>
00887 /// Save the document to a PDF stream.
00888 /// </summary>
00889 /// <param name="document">The <see cref="Document"/> to save.</param>
00890 /// <param name="stream">The stream to which the PDF data will be written.</param>
00891 /// <param name="textOption">Defines whether the used fonts should be included in the file.</param>
00892 /// <param name="compressStreams">Indicates whether the streams in the PDF file should be
      compressed.</param>
00893 /// <param name="linkDestinations">A dictionary associating element tags to link targets.  If this is
      provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to
      the destination specified in the dictionary.  If the destination starts with a hash (#), it is
      interpreted as the tag of another object in the current document; otherwise, it is interpreted as an
      external URI.</param>
00894 /// <param name="outline">Document outline (table of contents).</param>
00895 /// <param name="metadata">Document metadata.  Use <see cref="PDFMetadata.NoMetadata()"/> if you do
      not wish to include metadata in the document.</param>
00896 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
00897 /// <param name="optionalContentGroupSettings">Settings for optional content groups (layers).</param>
00898 /// <param name="annotationStyles">Annotation appearance styles.</param>
00899         public static void SaveAsPDF(this Document document, Stream stream, TextOptions textOption =
      TextOptions.SubsetFonts, bool compressStreams = true, Dictionary<string, string> linkDestinations =
      null, OutlineTree outline = null, PDFMetadata metadata = default, FilterOption filterOption = default,
      OptionalContentGroupSettings optionalContentGroupSettings = default, AnnotationStyleCollection
      annotationStyles = default)
00900         {
00901             PDFDocument pdfDoc = document.CreatePDFDocument(textOption, compressStreams,
      linkDestinations, outline, metadata, filterOption, optionalContentGroupSettings, annotationStyles);
00902             pdfDoc.Write(stream);
00903         }
00904     }
00905 }
```

8.33 PDFDocument.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2024 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
```

```
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.IO;
00021 using System.Text;
00022 using VectSharp.PDF.PDFObjects;
00023
00024 namespace VectSharp.PDF
00025 {
00026 /// <summary>
00027 /// A low-level representation of a PDF document.
00028 /// </summary>
00029     public class PDFDocument
00030     {
00031 /// <summary>
00032 /// Objects contained in the PDF document.
00033 /// </summary>
00034         public IList<PDFReferenceableObject> Contents { get; }
00035
00036 /// <summary>
00037 /// PDF specification version.  This is only included in the header and not enforced.
00038 /// </summary>
00039         public string PDFVersion { get; set; } = "1.4";
00040
00041 /// <summary>
00042 /// Gets or sets the PDF catalog object.
00043 /// </summary>
00044         public PDFCatalog Catalog
00045         {
00046             get
00047             {
00048                 bool found = false;
00049                 PDFCatalog tbr = null;
00050
00051                 foreach (PDFReferenceableObject obj in Contents)
00052                 {
00053                     if (obj is PDFCatalog cat)
00054                     {
00055                         if (!found)
00056                         {
00057                             found = true;
00058                             tbr = cat;
00059                         }
00060                         else
00061                         {
00062                             throw new ArgumentException("The document contains multiple catalog
      objects!");
00063                         }
00064                     }
00065                 }
00066
00067                 if (found)
00068                 {
00069                     return tbr;
00070                 }
00071                 else
00072                 {
00073                     throw new ArgumentOutOfRangeException("The document does not contain a catalog
      object!");
00074                 }
00075             }
00076
00077             set
00078             {
00079                 bool found = false;
00080                 int catalogIndex = -1;
00081
00082                 for (int i = 0; i < Contents.Count; i++)
00083                 {
00084                     if (Contents[i] is PDFCatalog)
00085                     {
00086                         if (!found)
00087                         {
00088                             found = true;
00089                             catalogIndex = i;
00090                         }
00091                         else
00092                         {
00093                             throw new ArgumentException("The document contains multiple catalog
      objects!");
00094                         }
00095                     }
00096                 }
00097
00098                 if (found)
00099                 {
00100                     Contents[catalogIndex] = value;
```

```
00101                 }
00102                 else
00103                 {
00104                     throw new ArgumentOutOfRangeException("The document does not contain a catalog
      object!");
00105                 }
00106             }
00107         }
00108 
00109 /// <summary>
00110 /// Gets or sets the PDF document info object.
00111 /// </summary>
00112         public PDFDocumentInfo Info
00113         {
00114             get
00115             {
00116                 bool found = false;
00117                 PDFDocumentInfo tbr = null;
00118 
00119                 foreach (PDFReferenceableObject obj in Contents)
00120                 {
00121                     if (obj is PDFDocumentInfo info)
00122                     {
00123                         if (!found)
00124                         {
00125                             found = true;
00126                             tbr = info;
00127                         }
00128                         else
00129                         {
00130                             throw new ArgumentException("The document contains multiple document info
      objects!");
00131                         }
00132                     }
00133                 }
00134 
00135                 if (found)
00136                 {
00137                     return tbr;
00138                 }
00139                 else
00140                 {
00141                     return null;
00142                 }
00143             }
00144 
00145             set
00146             {
00147                 bool found = false;
00148                 int infoIndex = -1;
00149 
00150                 for (int i = 0; i < Contents.Count; i++)
00151                 {
00152                     if (Contents[i] is PDFDocumentInfo)
00153                     {
00154                         if (!found)
00155                         {
00156                             found = true;
00157                             infoIndex = i;
00158                         }
00159                         else
00160                         {
00161                             throw new ArgumentException("The document contains multiple document info
      objects!");
00162                         }
00163                     }
00164                 }
00165                 if (value != null)
00166                 {
00167                     if (found)
00168                     {
00169                         Contents[infoIndex] = value;
00170                     }
00171                     else
00172                     {
00173                         Contents.Add(value);
00174                     }
00175                 }
00176                 else if (found)
00177                 {
00178                     Contents.RemoveAt(infoIndex);
00179                 }
00180             }
00181         }
00182 
00183 /// <summary>
00184 /// Gets the document page list.
```

```
00185 /// </summary>
00186         public List<PDFPage> Pages
00187         {
00188             get => this.Catalog.Pages.Kids.Values;
00189         }
00190 
00191 /// <summary>
00192 /// Default document encoding.
00193 /// </summary>
00194         public static readonly Encoding DefaultEncoding = Encoding.GetEncoding("ISO-8859-1");
00195 
00196 /// <summary>
00197 /// Write the PDF document to the specified <see cref="Stream"/>.
00198 /// </summary>
00199 /// <param name="stream">The stream on which the PDF document should be written.</param>
00200 /// <param name="encoding">The document encoding (default:  ISO 8859-1).</param>
00201         public void Write(Stream stream, Encoding encoding = default)
00202         {
00203             if (encoding == null)
00204             {
00205                 encoding = DefaultEncoding;
00206             }
00207 
00208             for (int i = 0; i < this.Contents.Count; i++)
00209             {
00210                 this.Contents[i].ObjectNumber = i + 1;
00211             }
00212 
00213             long[] objectPositions = new long[this.Contents.Count];
00214 
00215             StreamWriter sw = new StreamWriter(stream, encoding, 1024, true);
00216 
00217             //Header
00218             sw.Write("%PDF-" + this.PDFVersion + "\n");
00219             sw.Flush();
00220 
00221             for (int i = 0; i < this.Contents.Count; i++)
00222             {
00223                 objectPositions[i] = stream.Position;
00224                 sw.Write(this.Contents[i].ObjectNumber.ToString() + " " +
      this.Contents[i].Generation.ToString() + " obj\n");
00225                 sw.Flush();
00226                 this.Contents[i].FullWrite(stream, sw);
00227                 sw.Write("\nendobj\n");
00228                 sw.Flush();
00229             }
00230 
00231             long startXref = stream.Position;
00232 
00233             //XRef
00234             sw.Write("\nxref\n0 " + (objectPositions.Length + 1).ToString() + "\n0000000000 65535 f
      \n");
00235             for (int i = 0; i < objectPositions.Length; i++)
00236             {
00237                 sw.Write(objectPositions[i].ToString("0000000000",
      System.Globalization.CultureInfo.InvariantCulture) + " 00000 n \n");
00238             }
00239 
00240             //Trailer
00241 
00242             sw.Write("trailer\n");
00243             sw.Flush();
00244             PDFTrailer trailer = new PDFTrailer(objectPositions.Length + 1, this.Catalog, this.Info);
00245             trailer.Write(stream, sw);
00246 
00247             sw.Write("\nstartxref\n" + startXref.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + "\n%%EOF\n");
00248 
00249             sw.Flush();
00250             sw.Dispose();
00251         }
00252 
00253 /// <summary>
00254 /// Write the PDF document to the specified file.
00255 /// </summary>
00256 /// <param name="file">The file on which the PDF document should be written.</param>
00257 /// <param name="encoding">The document encoding (default:  ISO 8859-1).</param>
00258         public void Write(string file, Encoding encoding = default)
00259         {
00260             using (FileStream fs = new FileStream(file, FileMode.Create))
00261             {
00262                 this.Write(fs, encoding);
00263             }
00264         }
00265 
00266 /// <summary>
00267 /// Create a new <see cref="PDFDocument"/> containing the specified <paramref name="pdfObjects"/>.
```

```
00268 /// </summary>
00269 /// <param name="pdfObjects">The objects contained in the PDF document.</param>
00270         public PDFDocument(IList<PDFReferenceableObject> pdfObjects)
00271         {
00272             this.Contents = pdfObjects;
00273         }
00274     }
00275 }
```

8.34 PDFMetadata.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2024 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using VectSharp.PDF.PDFObjects;
00021
00022 namespace VectSharp.PDF
00023 {
00024 /// <summary>
00025 /// Represents metadata for a PDF document.
00026 /// </summary>
00027     public class PDFMetadata
00028     {
00029 /// <summary>
00030 /// Indicates that no metadata should be included in PDF document.
00031 /// </summary>
00032         public bool ExcludeMetadata { get; set; } = false;
00033 /// <summary>
00034 /// The document title.
00035 /// </summary>
00036         public string Title { get; set; }
00037
00038 /// <summary>
00039 /// The person who created the document.
00040 /// </summary>
00041         public string Author { get; set; } = Environment.UserName;
00042
00043 /// <summary>
00044 /// The subject of the document.
00045 /// </summary>
00046         public string Subject { get; set; }
00047
00048 /// <summary>
00049 /// Keywords associated with the document.
00050 /// </summary>
00051         public string Keywords { get; set; }
00052
00053 /// <summary>
00054 /// The program used to create the original version of the document (e.g., VectSharp).
00055 /// </summary>
00056         public string Creator { get; set; } = "VectSharp v" +
      typeof(Document).Assembly.GetName().Version.ToString(3);
00057
00058 /// <summary>
00059 /// The program used to create the PDF document (e.g., VectSharp.PDF).
00060 /// </summary>
00061         public string Producer { get; set; } = "VectSharp.PDF v" +
      typeof(PDFMetadata).Assembly.GetName().Version.ToString(3);
00062
00063 /// <summary>
00064 /// The date when the PDF file was created.
00065 /// </summary>
00066         public DateTime?  CreationDate { get; set; } = DateTime.Now;
00067
00068 /// <summary>
00069 /// The time zone where the PDF file was created.
00070 /// </summary>
```

```
00071         public TimeZoneInfo CreationDateTimeZone { get; set; } = TimeZoneInfo.Local;
00072
00073 /// <summary>
00074 /// The date when the PDF file was last modified.
00075 /// </summary>
00076         public DateTime?  ModificationDate { get; set; } = DateTime.Now;
00077
00078 /// <summary>
00079 /// The time zone where the PDF file was last modified.
00080 /// </summary>
00081         public TimeZoneInfo ModificationDateTimeZone { get; set; } = TimeZoneInfo.Local;
00082
00083 /// <summary>
00084 /// Custom metadata properties to include in the PDF document.
00085 /// </summary>
00086         public Dictionary<string, object> CustomProperties { get; set; }
00087
00088 /// <summary>
00089 /// Create a new PDF metadata object.
00090 /// </summary>
00091         public PDFMetadata()
00092         {
00093
00094         }
00095
00096         internal PDFDocumentInfo ToPDFDocumentInfo()
00097         {
00098             if (this.ExcludeMetadata)
00099             {
00100                 return null;
00101             }
00102             else
00103             {
00104                 PDFDocumentInfo tbr = new PDFDocumentInfo();
00105
00106                 if (!string.IsNullOrEmpty(Title))
00107                 {
00108                     tbr.Title = new PDFString(this.Title, PDFString.StringDelimiter.Brackets);
00109                 }
00110
00111                 if (!string.IsNullOrEmpty(Author))
00112                 {
00113                     tbr.Author = new PDFString(this.Author, PDFString.StringDelimiter.Brackets);
00114                 }
00115
00116                 if (!string.IsNullOrEmpty(Subject))
00117                 {
00118                     tbr.Subject = new PDFString(this.Subject, PDFString.StringDelimiter.Brackets);
00119                 }
00120
00121                 if (!string.IsNullOrEmpty(Keywords))
00122                 {
00123                     tbr.Keywords = new PDFString(this.Keywords, PDFString.StringDelimiter.Brackets);
00124                 }
00125
00126                 if (!string.IsNullOrEmpty(Creator))
00127                 {
00128                     tbr.Creator = new PDFString(this.Creator, PDFString.StringDelimiter.Brackets);
00129                 }
00130
00131                 if (!string.IsNullOrEmpty(Producer))
00132                 {
00133                     tbr.Producer = new PDFString(this.Producer, PDFString.StringDelimiter.Brackets);
00134                 }
00135
00136                 if (this.CreationDate != null)
00137                 {
00138                     tbr.CreationDate = new PDFDate(this.CreationDate.Value, this.CreationDateTimeZone
      ??  TimeZoneInfo.Utc);
00139                 }
00140
00141                 if (this.ModificationDate != null)
00142                 {
00143                     tbr.ModDate = new PDFDate(this.ModificationDate.Value,
      this.ModificationDateTimeZone ??  TimeZoneInfo.Utc);
00144                 }
00145
00146                 if (this.CustomProperties != null)
00147                 {
00148                     foreach (KeyValuePair<string, object> kvp in this.CustomProperties)
00149                     {
00150                         if (kvp.Value != null)
00151                         {
00152                             if (kvp.Value is string s)
00153                             {
00154                                 tbr.CustomProperties[kvp.Key] = new PDFString(s,
      PDFString.StringDelimiter.Brackets);
```

```
00155                           }
00156                           else if (kvp.Value is double d)
00157                           {
00158                               tbr.CustomProperties[kvp.Key] = new
      PDFString(d.ToString(System.Globalization.CultureInfo.InvariantCulture),
      PDFString.StringDelimiter.Brackets);
00159                           }
00160                           else if (kvp.Value is DateTime t)
00161                           {
00162                               tbr.CustomProperties[kvp.Key] = new PDFDate(t, TimeZoneInfo.Utc);
00163                           }
00164                           else
00165                           {
00166                               tbr.CustomProperties[kvp.Key] = new PDFString(kvp.Value.ToString(),
      PDFString.StringDelimiter.Brackets);
00167                           }
00168                       }
00169                   }
00170               }
00171
00172               return tbr;
00173           }
00174       }
00175
00176 /// <summary>
00177 /// Returns an object that does not include any metadata in the PDF file.
00178 /// </summary>
00179 /// <returns>An object that does not include any metadata in the PDF file.</returns>
00180       public static PDFMetadata NoMetadata()
00181       {
00182           return new PDFMetadata() { ExcludeMetadata = true };
00183       }
00184   }
00185 }
```

8.35 PDFProcessFigure.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2024 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.IO;
00021 using System.Linq;
00022 using System.Text;
00023 using VectSharp.PDF.Figures;
00024 using VectSharp.PDF.PDFObjects;
00025
00026 namespace VectSharp.PDF
00027 {
00028     internal static class PDFProcessFigure
00029     {
00030         public static Rectangle MeasureFigure(IFigure figure, ref double[,] transformationMatrix,
      Stack<double[,]> savedStates)
00031         {
00032             if (figure is PathFigure)
00033             {
00034                 PathFigure fig = figure as PathFigure;
00035
00036                 double minX = double.MaxValue;
00037                 double maxX = double.MinValue;
00038                 double minY = double.MaxValue;
00039                 double maxY = double.MinValue;
00040
00041                 for (int i = 0; i < fig.Segments.Length; i++)
00042                 {
00043                     switch (fig.Segments[i].Type)
00044                     {
00045                         case Figures.SegmentType.Move:
```

```
00046                               {
00047                                   Point pt = transformationMatrix.Multiply(fig.Segments[i].Point);
00048                                   minX = Math.Min(minX, pt.X);
00049                                   minY = Math.Min(minY, pt.Y);
00050                                   maxX = Math.Max(maxX, pt.X);
00051                                   maxY = Math.Max(maxY, pt.Y);
00052                               }
00053                               break;
00054                           case Figures.SegmentType.Line:
00055                               {
00056                                   Point pt = transformationMatrix.Multiply(fig.Segments[i].Point);
00057                                   minX = Math.Min(minX, pt.X);
00058                                   minY = Math.Min(minY, pt.Y);
00059                                   maxX = Math.Max(maxX, pt.X);
00060                                   maxY = Math.Max(maxY, pt.Y);
00061                               }
00062                               break;
00063                           case Figures.SegmentType.CubicBezier:
00064                               for (int j = 0; j < fig.Segments[i].Points.Length; j++)
00065                               {
00066                                   Point pt = transformationMatrix.Multiply(fig.Segments[i].Points[j]);
00067                                   minX = Math.Min(minX, pt.X);
00068                                   minY = Math.Min(minY, pt.Y);
00069                                   maxX = Math.Max(maxX, pt.X);
00070                                   maxY = Math.Max(maxY, pt.Y);
00071                               }
00072                               break;
00073                           case Figures.SegmentType.Close:
00074                               break;
00075                       }
00076                   }
00077
00078                   return new Rectangle(minX, minY, maxX - minX, maxY - minY);
00079               }
00080               else if (figure is TextFigure)
00081               {
00082                   TextFigure fig = figure as TextFigure;
00083
00084                   double yMax = 0;
00085                   double yMin = 0;
00086
00087                   if (fig.Font.FontFamily.TrueTypeFile != null)
00088                   {
00089                       for (int i = 0; i < fig.Text.Length; i++)
00090                       {
00091                           TrueTypeFile.VerticalMetrics vMet =
      fig.Font.FontFamily.TrueTypeFile.Get1000EmGlyphVerticalMetrics(fig.Text[i]);
00092                           yMin = Math.Min(yMin, vMet.YMin * fig.Font.FontSize / 1000);
00093                           yMax = Math.Max(yMax, vMet.YMax * fig.Font.FontSize / 1000);
00094                       }
00095                   }
00096
00097                   double realY = fig.Position.Y;
00098
00099                   if (fig.TextBaseline == TextBaselines.Bottom)
00100                   {
00101                       realY -= yMax;
00102                   }
00103                   else if (fig.TextBaseline == TextBaselines.Top)
00104                   {
00105                       realY -= yMin;
00106                   }
00107                   else if (fig.TextBaseline == TextBaselines.Middle)
00108                   {
00109                       realY -= (yMax + yMin) * 0.5;
00110                   }
00111                   else if (fig.TextBaseline == TextBaselines.Baseline)
00112                   {
00113                       realY -= yMax + yMin;
00114                   }
00115
00116                   Font.DetailedFontMetrics metrics = fig.Font.MeasureTextAdvanced(fig.Text);
00117
00118                   Point corner1 = new Point(fig.Position.X - metrics.LeftSideBearing, realY +
      metrics.Top);
00119                   Point corner2 = new Point(fig.Position.X + metrics.Width, realY + metrics.Top);
00120                   Point corner3 = new Point(fig.Position.X + metrics.Width, realY + metrics.Bottom);
00121                   Point corner4 = new Point(fig.Position.X - metrics.LeftSideBearing, realY +
      metrics.Bottom);
00122
00123                   corner1 = transformationMatrix.Multiply(corner1);
00124                   corner2 = transformationMatrix.Multiply(corner2);
00125                   corner3 = transformationMatrix.Multiply(corner3);
00126                   corner4 = transformationMatrix.Multiply(corner4);
00127
00128                   double x = Math.Min(corner1.X, Math.Min(corner2.X, Math.Min(corner3.X, corner4.X)));
00129                   double y = Math.Min(corner1.Y, Math.Min(corner2.Y, Math.Min(corner3.Y, corner4.Y)));
```

```
00130                 double w = Math.Max(corner1.X, Math.Max(corner2.X, Math.Max(corner3.X, corner4.X))) -
      x;
00131                 double h = Math.Max(corner1.Y, Math.Max(corner2.Y, Math.Max(corner3.Y, corner4.Y))) -
      y;
00132
00133                 return new Rectangle(x, y, w, h);
00134             }
00135             else if (figure is TransformFigure transf)
00136             {
00137                 if (transf.TransformType == TransformFigure.TransformTypes.Transform)
00138                 {
00139                     transformationMatrix = transformationMatrix.Multiply(transf.TransformationMatrix);
00140                 }
00141                 else if (transf.TransformType == TransformFigure.TransformTypes.Save)
00142                 {
00143                     savedStates.Push(transformationMatrix);
00144                 }
00145                 else if (transf.TransformType == TransformFigure.TransformTypes.Restore)
00146                 {
00147                     transformationMatrix = savedStates.Pop();
00148                 }
00149
00150                 return new Rectangle(0, 0, 0, 0);
00151             }
00152             else if (figure is RasterImageFigure)
00153             {
00154                 Point corner1 = new Point(0, 0);
00155                 Point corner2 = new Point(0, 1);
00156                 Point corner3 = new Point(1, 1);
00157                 Point corner4 = new Point(1, 0);
00158
00159                 corner1 = transformationMatrix.Multiply(corner1);
00160                 corner2 = transformationMatrix.Multiply(corner2);
00161                 corner3 = transformationMatrix.Multiply(corner3);
00162                 corner4 = transformationMatrix.Multiply(corner4);
00163
00164                 double x = Math.Min(corner1.X, Math.Min(corner2.X, Math.Min(corner3.X, corner4.X)));
00165                 double y = Math.Min(corner1.Y, Math.Min(corner2.Y, Math.Min(corner3.Y, corner4.Y)));
00166                 double w = Math.Max(corner1.X, Math.Max(corner2.X, Math.Max(corner3.X, corner4.X))) -
      x;
00167                 double h = Math.Max(corner1.Y, Math.Max(corner2.Y, Math.Max(corner3.Y, corner4.Y))) -
      y;
00168
00169                 return new Rectangle(x, y, w, h);
00170             }
00171             else
00172             {
00173                 return new Rectangle(0, 0, 0, 0);
00174             }
00175         }
00176
00177         private static double[,] Multiply(this double[,] matrix, double[,] matrix2)
00178         {
00179             double[,] tbr = new double[3, 3];
00180
00181             tbr[0, 0] = matrix[0, 0] * matrix2[0, 0] - matrix[0, 1] * matrix2[1, 0] + matrix[0, 2] *
      matrix2[2, 0];
00182             tbr[0, 1] = -matrix[0, 0] * matrix2[0, 1] + matrix[0, 1] * matrix2[1, 1] + matrix[0, 2] *
      matrix2[2, 1];
00183             tbr[0, 2] = matrix[0, 0] * matrix2[0, 2] + matrix[0, 1] * matrix2[1, 2] + matrix[0, 2] *
      matrix2[2, 2];
00184
00185             tbr[1, 0] = matrix[1, 0] * matrix2[0, 0] - matrix[1, 1] * matrix2[1, 0] + matrix[1, 2] *
      matrix2[2, 0];
00186             tbr[1, 1] = -matrix[1, 0] * matrix2[0, 1] + matrix[1, 1] * matrix2[1, 1] + matrix[1, 2] *
      matrix2[2, 1];
00187             tbr[1, 2] = matrix[1, 0] * matrix2[0, 2] + matrix[1, 1] * matrix2[1, 2] + matrix[1, 2] *
      matrix2[2, 2];
00188
00189             tbr[2, 0] = matrix[2, 0] * matrix2[0, 0] - matrix[2, 1] * matrix2[1, 0] + matrix[2, 2] *
      matrix2[2, 0];
00190             tbr[2, 1] = -matrix[2, 0] * matrix2[0, 1] + matrix[2, 1] * matrix2[1, 1] + matrix[2, 2] *
      matrix2[2, 1];
00191             tbr[2, 2] = matrix[2, 0] * matrix2[0, 2] + matrix[2, 1] * matrix2[1, 2] + matrix[2, 2] *
      matrix2[2, 2];
00192
00193             return tbr;
00194         }
00195
00196         private static Point Multiply(this double[,] matrix, Point point)
00197         {
00198             double[] transPt = new double[3];
00199
00200             transPt[0] = matrix[0, 0] * point.X + matrix[0, 1] * point.Y + matrix[0, 2];
00201             transPt[1] = matrix[1, 0] * point.X + matrix[1, 1] * point.Y + matrix[1, 2];
00202             transPt[2] = matrix[2, 0] * point.X + matrix[2, 1] * point.Y + matrix[2, 2];
00203
```

```
00204             return new Point(transPt[0] / transPt[2], transPt[1] / transPt[2]);
00205         }
00206
00207         private static string GetKernedString(string text, Font font)
00208         {
00209             List<(string, Point)> tSpans = new List<(string, Point)>();
00210
00211             StringBuilder currentRun = new StringBuilder();
00212             Point currentKerning = new Point();
00213
00214             Point currentGlyphPlacementDelta = new Point();
00215             Point currentGlyphAdvanceDelta = new Point();
00216             Point nextGlyphPlacementDelta = new Point();
00217             Point nextGlyphAdvanceDelta = new Point();
00218
00219             for (int i = 0; i < text.Length; i++)
00220             {
00221                 if (i < text.Length - 1)
00222                 {
00223                     currentGlyphPlacementDelta = nextGlyphPlacementDelta;
00224                     currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
00225                     nextGlyphAdvanceDelta = new Point();
00226                     nextGlyphPlacementDelta = new Point();
00227
00228                     TrueTypeFile.PairKerning kerning =
    font.FontFamily.TrueTypeFile.Get1000EmKerning(text[i], text[i + 1]);
00229
00230                     if (kerning != null)
00231                     {
00232                         currentGlyphPlacementDelta = new Point(currentGlyphPlacementDelta.X +
    kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y + kerning.Glyph1Placement.Y);
00233                         currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
    kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
00234
00235                         nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
    kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
00236                         nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
    kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
00237                     }
00238                 }
00239
00240                 if (currentGlyphPlacementDelta.X != 0 || currentGlyphPlacementDelta.Y != 0 ||
    currentGlyphAdvanceDelta.X != 0 || currentGlyphAdvanceDelta.Y != 0)
00241                 {
00242                     if (currentRun.Length > 0)
00243                     {
00244                         tSpans.Add((currentRun.ToString(), currentKerning));
00245
00246                         tSpans.Add((text[i].ToString(), new Point(currentGlyphPlacementDelta.X,
    currentGlyphPlacementDelta.Y)));
00247
00248                         currentRun.Clear();
00249                         currentKerning = new Point(currentGlyphAdvanceDelta.X -
    currentGlyphPlacementDelta.X, currentGlyphAdvanceDelta.Y - currentGlyphPlacementDelta.Y);
00250                     }
00251                     else
00252                     {
00253                         tSpans.Add((text[i].ToString(), new Point(currentGlyphPlacementDelta.X +
    currentKerning.X, currentGlyphPlacementDelta.Y + currentKerning.Y)));
00254
00255                         currentRun.Clear();
00256                         currentKerning = new Point(currentGlyphAdvanceDelta.X -
    currentGlyphPlacementDelta.X, currentGlyphAdvanceDelta.Y - currentGlyphPlacementDelta.Y);
00257                     }
00258                 }
00259                 else
00260                 {
00261                     currentRun.Append(text[i]);
00262                 }
00263             }
00264
00265             if (currentRun.Length > 0)
00266             {
00267                 tSpans.Add((currentRun.ToString(), currentKerning));
00268             }
00269
00270             StringBuilder sb = new StringBuilder();
00271             sb.Append("[");
00272
00273             for (int i = 0; i < tSpans.Count; i++)
00274             {
00275                 if (tSpans[i].Item2.X != 0)
00276                 {
00277                     sb.Append((-tSpans[i].Item2.X).ToString("0.################",
    System.Globalization.CultureInfo.InvariantCulture));
00278                 }
00279
```

```
00280                 sb.Append("(");
00281                 sb.Append(PDFString.EscapeStringForPDF(tSpans[i].Item1));
00282                 sb.Append(")");
00283             }
00284 
00285             sb.Append("]");
00286 
00287             return sb.ToString();
00288         }
00289 
00290         private static string EscapeSymbolStringForPDF(string str, Dictionary<char, int> glyphIndices)
00291         {
00292             StringBuilder sb = new StringBuilder();
00293 
00294             for (int i = 0; i < str.Length; i++)
00295             {
00296                 sb.Append((glyphIndices[str[i]]).ToString("X4"));
00297             }
00298             return sb.ToString();
00299         }
00300 
00301         public static void WriteFigure(IFigure figure, Dictionary<(string fontFamilyName, bool
      isSymbolic), PDFFont> fonts, double[] alphas, Dictionary<string, PDFImage> imageObjects, double[,]
      transformationMatrix, List<(GradientBrush, double[,], IFigure)> gradients, Dictionary<string,
      PDFOptionalContentGroupMembership> optionalContentGroups, StreamWriter sw)
00302         {
00303             bool restoreAtEnd = false;
00304 
00305             if (figure.Fill != null)
00306             {
00307                 if (figure.Fill is SolidColourBrush solid)
00308                 {
00309                     sw.Write(solid.R.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " + solid.G.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " + solid.B.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " rg\n");
00310                     sw.Write("/a" + Array.IndexOf(alphas, solid.A).ToString() + " gs\n");
00311                 }
00312                 else if (figure.Fill is GradientBrush gradient)
00313                 {
00314                     int brushIndex = gradients.Count;
00315                     double[,] clonedMatrix = new double[3, 3] { { transformationMatrix[0, 0],
      transformationMatrix[0, 1], transformationMatrix[0, 2] }, { transformationMatrix[1, 0],
      transformationMatrix[1, 1], transformationMatrix[1, 2] }, { transformationMatrix[2, 0],
      transformationMatrix[2, 1], transformationMatrix[2, 2] } };
00316 
00317                     gradients.Add((gradient, clonedMatrix, figure));
00318 
00319                     bool gradientCompatible = PDFContext.IsCompatible(gradient);
00320 
00321                     if (!gradientCompatible)
00322                     {
00323                         sw.Write("q\n");
00324                         sw.Write("/ma" +
      brushIndex.ToString(System.Globalization.CultureInfo.InvariantCulture) + " gs ");
00325                         restoreAtEnd = true;
00326                     }
00327 
00328                     sw.Write("/Pattern cs /p" +
      brushIndex.ToString(System.Globalization.CultureInfo.InvariantCulture) + " scn /a" +
      Array.IndexOf(alphas, 1.0).ToString() + " gs\n");
00329                 }
00330             }
00331 
00332             if (figure.Stroke != null)
00333             {
00334                 if (figure.Stroke is SolidColourBrush solid)
00335                 {
00336                     sw.Write(solid.R.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " + solid.G.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " + solid.B.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " RG\n");
00337                     sw.Write("/a" + Array.IndexOf(alphas, solid.A).ToString() + " gs\n");
00338                 }
00339                 else if (figure.Stroke is GradientBrush gradient)
00340                 {
00341                     int brushIndex = gradients.Count;
00342                     double[,] clonedMatrix = new double[3, 3] { { transformationMatrix[0, 0],
      transformationMatrix[0, 1], transformationMatrix[0, 2] }, { transformationMatrix[1, 0],
      transformationMatrix[1, 1], transformationMatrix[1, 2] }, { transformationMatrix[2, 0],
      transformationMatrix[2, 1], transformationMatrix[2, 2] } };
00343 
00344                     gradients.Add((gradient, clonedMatrix, figure));
00345 
00346                     bool gradientCompatible = PDFContext.IsCompatible(gradient);
00347 
00348                     if (!gradientCompatible)
```

```
00349                     {
00350                         sw.Write("q\n");
00351                         sw.Write("/ma" +
      brushIndex.ToString(System.Globalization.CultureInfo.InvariantCulture) + " gs ");
00352                         restoreAtEnd = true;
00353                     }
00354 
00355                     sw.Write("/Pattern CS /p" +
      brushIndex.ToString(System.Globalization.CultureInfo.InvariantCulture) + " SCN /a" +
      Array.IndexOf(alphas, 1.0).ToString() + " gs\n");
00356                 }
00357 
00358                 sw.Write(figure.LineWidth.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " w\n");
00359                 sw.Write(((int)figure.LineCap).ToString() + " J\n");
00360                 sw.Write(((int)figure.LineJoin).ToString() + " j\n");
00361                 if (figure.LineDash.DashArray != null && figure.LineDash.DashArray.Any(x => x != 0))
00362                 {
00363                     sw.Write("[ ");
00364 
00365                     for (int i = 0; i < figure.LineDash.DashArray.Length; i++)
00366                     {
00367                         sw.Write(figure.LineDash.DashArray[i].ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture));
00368                         sw.Write(" ");
00369                     }
00370 
00371                     sw.Write("] " + figure.LineDash.Phase.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " d\n");
00372                 }
00373                 else
00374                 {
00375                     sw.Write("[] 0 d\n");
00376                 }
00377             }
00378 
00379             if (figure is PathFigure)
00380             {
00381                 PathFigure fig = figure as PathFigure;
00382 
00383                 for (int i = 0; i < fig.Segments.Length; i++)
00384                 {
00385                     switch (fig.Segments[i].Type)
00386                     {
00387                         case Figures.SegmentType.Move:
00388                             {
00389                                 sw.Write(fig.Segments[i].Point.X.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " +
      fig.Segments[i].Point.Y.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " m ");
00390                             }
00391                             break;
00392                         case Figures.SegmentType.Line:
00393                             {
00394                                 sw.Write(fig.Segments[i].Point.X.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " +
      fig.Segments[i].Point.Y.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " l ");
00395                             }
00396                             break;
00397                         case Figures.SegmentType.CubicBezier:
00398                             for (int j = 0; j < fig.Segments[i].Points.Length; j++)
00399                             {
00400                                 sw.Write(fig.Segments[i].Points[j].X.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " +
      fig.Segments[i].Points[j].Y.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " ");
00401                             }
00402                             sw.Write("c ");
00403                             break;
00404                         case Figures.SegmentType.Close:
00405                             sw.Write("h ");
00406                             break;
00407                     }
00408                 }
00409 
00410                 if (fig.IsClipping)
00411                 {
00412                     sw.Write("W n\n");
00413                 }
00414                 else
00415                 {
00416                     if (fig.Fill != null)
00417                     {
00418                         if (fig.FillRule == FillRule.EvenOdd)
00419                         {
00420                             sw.Write("f*\n");
```

```
00421                         }
00422                         else
00423                         {
00424                             sw.Write("f\n");
00425                         }
00426                     }
00427 
00428                     if (fig.Stroke != null)
00429                     {
00430                         sw.Write("S\n");
00431                     }
00432                 }
00433             }
00434             else if (figure is TextFigure)
00435             {
00436                 TextFigure fig = figure as TextFigure;
00437 
00438                 List<(string txt, bool isSymbolic)> segments = new List<(string txt, bool
      isSymbolic)>();
00439 
00440                 StringBuilder currSeg = new StringBuilder();
00441                 bool currSymbolic = false;
00442 
00443                 for (int i = 0; i < fig.Text.Length; i++)
00444                 {
00445                     if (PDFContextInterpreter.CP1252Chars.Contains(fig.Text[i]))
00446                     {
00447                         if (!currSymbolic)
00448                         {
00449                             currSeg.Append(fig.Text[i]);
00450                         }
00451                         else
00452                         {
00453                             if (currSeg.Length > 0)
00454                             {
00455                                 segments.Add((currSeg.ToString(), currSymbolic));
00456                             }
00457 
00458                             currSeg = new StringBuilder();
00459                             currSymbolic = false;
00460                             currSeg.Append(fig.Text[i]);
00461                         }
00462                     }
00463                     else
00464                     {
00465                         if (currSymbolic)
00466                         {
00467                             currSeg.Append(fig.Text[i]);
00468                         }
00469                         else
00470                         {
00471                             if (currSeg.Length > 0)
00472                             {
00473                                 segments.Add((currSeg.ToString(), currSymbolic));
00474                             }
00475 
00476                             currSeg = new StringBuilder();
00477                             currSymbolic = true;
00478                             currSeg.Append(fig.Text[i]);
00479                         }
00480                     }
00481                 }
00482 
00483                 if (currSeg.Length > 0)
00484                 {
00485                     segments.Add((currSeg.ToString(), currSymbolic));
00486                 }
00487 
00488 
00489 
00490                 double realX = fig.Position.X;
00491 
00492                 if (fig.Font.FontFamily.TrueTypeFile != null)
00493                 {
00494                     realX = fig.Position.X -
      fig.Font.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(fig.Text[0]).LeftSideBearing *
      fig.Font.FontSize / 1000;
00495                 }
00496 
00497                 double yMax = 0;
00498                 double yMin = 0;
00499 
00500                 if (fig.Font.FontFamily.TrueTypeFile != null)
00501                 {
00502                     for (int i = 0; i < fig.Text.Length; i++)
00503                     {
00504                         TrueTypeFile.VerticalMetrics vMet =
```

```
      fig.Font.FontFamily.TrueTypeFile.Get1000EmGlyphVerticalMetrics(fig.Text[i]);
00505                         yMin = Math.Min(yMin, vMet.YMin * fig.Font.FontSize / 1000);
00506                         yMax = Math.Max(yMax, vMet.YMax * fig.Font.FontSize / 1000);
00507                     }
00508                 }
00509
00510                 double realY = fig.Position.Y;
00511
00512                 if (fig.TextBaseline == TextBaselines.Bottom)
00513                 {
00514                     realY -= yMax;
00515                 }
00516                 else if (fig.TextBaseline == TextBaselines.Top)
00517                 {
00518                     realY -= yMin;
00519                 }
00520                 else if (fig.TextBaseline == TextBaselines.Middle)
00521                 {
00522                     realY -= (yMax + yMin) * 0.5;
00523                 }
00524                 else if (fig.TextBaseline == TextBaselines.Baseline)
00525                 {
00526                     realY -= yMax + yMin;
00527                 }
00528
00529                 double middleY = realY + (yMax + yMin) * 0.5;
00530
00531
00532
00533                 sw.Write("q\n1 0 0 1 0 " + (middleY).ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " cm\n");
00534                 sw.Write("1 0 0 -1 0 0 cm\n");
00535                 sw.Write("1 0 0 1 0 " + (-middleY).ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " cm\n");
00536
00537                 sw.Write("BT\n");
00538
00539                 if (figure.Stroke != null && figure.Fill != null)
00540                 {
00541                     sw.Write("2 Tr\n");
00542                 }
00543                 else if (figure.Stroke != null)
00544                 {
00545                     sw.Write("1 Tr\n");
00546                 }
00547                 else if (figure.Fill != null)
00548                 {
00549                     sw.Write("0 Tr\n");
00550                 }
00551
00552                 sw.Write(realX.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " " + realY.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " Td\n");
00553
00554                 for (int i = 0; i < segments.Count; i++)
00555                 {
00556                     if (!segments[i].isSymbolic)
00557                     {
00558                         sw.Write("/" + fonts[(fig.Font.FontFamily.FileName, false)].FontReferenceName
      + " " + fig.Font.FontSize.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " Tf\n");
00559                         sw.Write(GetKernedString(segments[i].txt, fig.Font) + " TJ\n");
00560                     }
00561                     else
00562                     {
00563                         sw.Write("/" + fonts[(fig.Font.FontFamily.FileName, true)].FontReferenceName +
      " " + fig.Font.FontSize.ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " Tf\n");
00564                         sw.Write("<" + EscapeSymbolStringForPDF(segments[i].txt,
      ((PDFType0Font)fonts[(fig.Font.FontFamily.FileName, true)]).DescendantFonts.Values[0].GlyphIndices) +
      "> Tj\n");
00565                     }
00566                 }
00567
00568                 sw.Write("ET\nQ\n");
00569             }
00570             else if (figure is TransformFigure transf)
00571             {
00572                 if (transf.TransformType == TransformFigure.TransformTypes.Transform)
00573                 {
00574                     sw.Write(transf.TransformationMatrix[0, 0].ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " ");
00575                     sw.Write(transf.TransformationMatrix[0, 1].ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " ");
00576                     sw.Write(transf.TransformationMatrix[1, 0].ToString("0.################",
      System.Globalization.CultureInfo.InvariantCulture) + " ");
00577                     sw.Write(transf.TransformationMatrix[1, 1].ToString("0.################",
```

```
System.Globalization.CultureInfo.InvariantCulture) + " ");
00578                     sw.Write(transf.TransformationMatrix[0, 2].ToString("0.################",
System.Globalization.CultureInfo.InvariantCulture) + " ");
00579                     sw.Write(transf.TransformationMatrix[1, 2].ToString("0.################",
System.Globalization.CultureInfo.InvariantCulture) + " cm\n");
00580                 }
00581                 else if (transf.TransformType == TransformFigure.TransformTypes.Save)
00582                 {
00583                     sw.Write("q\n");
00584                 }
00585                 else if (transf.TransformType == TransformFigure.TransformTypes.Restore)
00586                 {
00587                     sw.Write("Q\n");
00588                 }
00589             }
00590             else if (figure is RasterImageFigure fig)
00591             {
00592                 sw.Write("/a" + Array.IndexOf(alphas, 1).ToString() + " gs\n");
00593                 sw.Write("/" + imageObjects[fig.Image.Id].ReferenceName + " Do\n");
00594             }
00595             else if (figure is OptionalContentFigure opt)
00596             {
00597                 if (opt.FigureType == OptionalContentFigure.OptionalContentType.Start)
00598                 {
00599                     sw.Write("/OC /" +
optionalContentGroups[opt.VisibilityExpression.ToString()].ReferenceName + " BDC\n");
00600                 }
00601                 else if (opt.FigureType == OptionalContentFigure.OptionalContentType.End)
00602                 {
00603                     sw.Write("EMC\n");
00604                 }
00605             }
00606
00607             if (restoreAtEnd)
00608             {
00609                 sw.Write("Q\n");
00610             }
00611         }
00612     }
00613 }
```

8.36 Axes.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020
00021 namespace VectSharp.Plots
00022 {
00023 /// <summary>
00024 /// A plot element that draws an axis line and an arrow.
00025 /// </summary>
00026     public class ContinuousAxis :  IPlotElement
00027     {
00028 /// <summary>
00029 /// The starting point of the axis (i.e., the blunt end of the arrow), expressed in data space
coordinates.
00030 /// </summary>
00031         public IReadOnlyList<double> StartPoint { get; set; }
00032
00033 /// <summary>
00034 /// The ending point of the axis (i.e., the pointy end of the arrow), expressed in data space
coordinates.
00035 /// </summary>
00036         public IReadOnlyList<double> EndPoint { get; set; }
00037
00038 /// <summary>
```

```
00039 /// The size of the arrow at the end of the axis, expressed in plot space coordinates.
00040 /// </summary>
00041         public double ArrowSize { get; set; } = 3;
00042 
00043 /// <summary>
00044 /// The coordinate system used to transform the points from data space to plot space.
00045 /// </summary>
00046         public IContinuousCoordinateSystem CoordinateSystem { get; set; }
00047         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00048 
00049 /// <summary>
00050 /// Presentation attributes determining the appearance of the axis.
00051 /// </summary>
00052         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes();
00053 
00054 /// <summary>
00055 /// A tag to identify the axis in the plot.
00056 /// </summary>
00057         public string Tag { get; set; }
00058 
00059 /// <summary>
00060 /// Creates a new <see cref="ContinuousAxis"/> instance.
00061 /// </summary>
00062 /// <param name="startPoint">The starting point of the axis (i.e., the blunt end of the arrow),
      expressed in data space coordinates.</param>
00063 /// <param name="endPoint">The ending point of the axis (i.e., the pointy end of the arrow), expressed
      in data space coordinates.</param>
00064 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00065         public ContinuousAxis(IReadOnlyList<double> startPoint, IReadOnlyList<double> endPoint,
      IContinuousCoordinateSystem coordinateSystem)
00066         {
00067             this.StartPoint = startPoint;
00068             this.EndPoint = endPoint;
00069             this.CoordinateSystem = coordinateSystem;
00070         }
00071 
00072 /// <inheritdoc/>
00073         public void Plot(Graphics target)
00074         {
00075             Point startPoint = CoordinateSystem.ToPlotCoordinates(StartPoint);
00076             Point endPoint = CoordinateSystem.ToPlotCoordinates(EndPoint);
00077 
00078             double[] direction = new double[StartPoint.Count];
00079 
00080             for (int i = 0; i < direction.Length; i++)
00081             {
00082                 direction[i] = EndPoint[i] - StartPoint[i];
00083             }
00084 
00085             double angle;
00086 
00087             GraphicsPath path;
00088 
00089             if (CoordinateSystem.IsLinear || CoordinateSystem.IsDirectionStraight(direction))
00090             {
00091                 path = new GraphicsPath().MoveTo(startPoint).LineTo(endPoint);
00092                 angle = Math.Atan2(endPoint.Y - startPoint.Y, endPoint.X - startPoint.X);
00093             }
00094             else
00095             {
00096                 angle = 0;
00097 
00098                 path = new GraphicsPath();
00099 
00100                 int count = 0;
00101 
00102                 for (int i = 0; i < direction.Length; i++)
00103                 {
00104                     count = Math.Max(count, (int)Math.Ceiling(direction[i] /
      CoordinateSystem.Resolution[i]));
00105                 }
00106 
00107                 Point lastPoint = new Point();
00108 
00109                 for (int i = 0; i <= count; i++)
00110                 {
00111                     double[] pt = new double[StartPoint.Count];
00112 
00113                     for (int j = 0; j < pt.Length; j++)
00114                     {
00115                         pt[j] = StartPoint[j] + direction[j] / count * i;
00116                     }
00117 
00118                     Point point = CoordinateSystem.ToPlotCoordinates(pt);
00119                     path.LineTo(point);
```

```
00120
00121                     if (i == count)
00122                     {
00123                         angle = Math.Atan2(point.Y - lastPoint.Y, point.X - lastPoint.X);
00124                     }
00125                     else
00126                     {
00127                         lastPoint = point;
00128                     }
00129                 }
00130             }
00131
00132             target.StrokePath(path, PresentationAttributes.Stroke, PresentationAttributes.LineWidth,
      PresentationAttributes.LineCap, PresentationAttributes.LineJoin, PresentationAttributes.LineDash, tag:
      Tag);
00133
00134             if (ArrowSize > 0)
00135             {
00136                 target.Save();
00137                 target.Translate(endPoint);
00138
00139                 target.Rotate(angle);
00140
00141                 GraphicsPath arrowPath = new GraphicsPath().MoveTo(-ArrowSize *
      PresentationAttributes.LineWidth, -ArrowSize * PresentationAttributes.LineWidth).LineTo(ArrowSize *
      PresentationAttributes.LineWidth, 0).LineTo(-ArrowSize * PresentationAttributes.LineWidth, ArrowSize *
      PresentationAttributes.LineWidth).Close();
00142
00143                 string fillTag = Tag;
00144                 string strokeTag = Tag;
00145
00146                 if (target.UseUniqueTags && !string.IsNullOrEmpty(Tag))
00147                 {
00148                     fillTag += "@arrowFill";
00149                     strokeTag += "@arrowStroke";
00150                 }
00151
00152                 if (PresentationAttributes.Fill != null)
00153                 {
00154                     target.FillPath(arrowPath, PresentationAttributes.Fill, tag:  fillTag);
00155                 }
00156
00157                 target.StrokePath(arrowPath, PresentationAttributes.Stroke,
      PresentationAttributes.LineWidth, PresentationAttributes.LineCap, PresentationAttributes.LineJoin,
      PresentationAttributes.LineDash, tag:  strokeTag);
00158
00159                 target.Restore();
00160             }
00161         }
00162     }
00163
00164 /// <summary>
00165 /// A plot element that draws equally spaced ticks on an axis.
00166 /// </summary>
00167     public class ContinuousAxisTicks :  IPlotElement
00168     {
00169 /// <summary>
00170 /// The starting point of the axis, expressed in data space coordinates.
00171 /// </summary>
00172         public IReadOnlyList<double> StartPoint { get; set; }
00173
00174 /// <summary>
00175 /// The ending point of the axis, expressed in data space coordinates.
00176 /// </summary>
00177         public IReadOnlyList<double> EndPoint { get; set; }
00178
00179 /// <summary>
00180 /// The number of intervals between ticks.  Note that the number of ticks will be one greater than
      this.
00181 /// </summary>
00182         public double IntervalCount { get; set; } = 10;
00183
00184 /// <summary>
00185 /// The size of the ticks "above" the axis.  What "above" means depends on the orientation of the
      axis.
00186 /// This should be set to a function accepting an <see langword="int"/> argument representing the
      index of the tick, and
00187 /// return a <see langword="double"/> representing the size of the tick in plot coordinates.
00188 /// </summary>
00189         public Func<int, double> SizeAbove { get; set; } = i => i % 2 == 0 ?  3 :  2;
00190
00191 /// <summary>
00192 /// The size of the ticks "below" the axis.  What "below" means depends on the orientation of the
      axis.
00193 /// This should be set to a function accepting an <see langword="int"/> argument representing the
      index of the tick, and
00194 /// return a <see langword="double"/> representing the size of the tick in plot coordinates.
```

```
00195 /// </summary>
00196         public Func<int, double> SizeBelow { get; set; } = i => i % 2 == 0 ?  3 :  2;
00197 
00198 /// <summary>
00199 /// The coordinate system used to transform the points from data space to plot space.
00200 /// </summary>
00201         public IContinuousCoordinateSystem CoordinateSystem { get; set; }
00202         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00203 
00204 /// <summary>
00205 /// Presentation attributes determining the appearance of the ticks.
00206 /// </summary>
00207         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes();
00208 
00209 /// <summary>
00210 /// A tag to identify the ticks in the plot.
00211 /// </summary>
00212         public string Tag { get; set; }
00213 
00214 /// <summary>
00215 /// Creates a new <see cref="ContinuousAxisTicks"/> instance.
00216 /// </summary>
00217 /// <param name="startPoint">The starting point of the axis, expressed in data space
      coordinates.</param>
00218 /// <param name="endPoint">The ending point of the axis, expressed in data space coordinates.</param>
00219 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00220         public ContinuousAxisTicks(IReadOnlyList<double> startPoint, IReadOnlyList<double> endPoint,
      IContinuousCoordinateSystem coordinateSystem)
00221         {
00222             this.StartPoint = startPoint;
00223             this.EndPoint = endPoint;
00224             this.CoordinateSystem = coordinateSystem;
00225         }
00226 
00227 /// <inheritdoc/>
00228         public void Plot(Graphics target)
00229         {
00230             double[] direction = new double[StartPoint.Count];
00231             double directionMod = 0;
00232 
00233             for (int i = 0; i < direction.Length; i++)
00234             {
00235                 direction[i] = EndPoint[i] - StartPoint[i];
00236                 directionMod += direction[i] * direction[i];
00237             }
00238 
00239             directionMod = Math.Sqrt(directionMod);
00240 
00241             GraphicsPath tickPath = new GraphicsPath();
00242 
00243             double[] normDir = new double[StartPoint.Count];
00244             double[] invDir = new double[StartPoint.Count];
00245 
00246             for (int i = 0; i < direction.Length; i++)
00247             {
00248                 normDir[i] = direction[i] / directionMod;
00249                 invDir[i] = -normDir[i];
00250             }
00251 
00252             for (int i = 0; i <= IntervalCount; i++)
00253             {
00254                 double[] pt = new double[StartPoint.Count];
00255 
00256                 if (CoordinateSystem is IContinuousInvertibleCoordinateSystem inv &&
      inv.IsDirectionStraight(direction))
00257                 {
00258                     Point start = inv.ToPlotCoordinates(StartPoint);
00259                     Point end = inv.ToPlotCoordinates(EndPoint);
00260 
00261                     if (IntervalCount != 0)
00262                     {
00263                         pt = inv.ToDataCoordinates(new Point(start.X * (1 - i / IntervalCount) + end.X
      * i / IntervalCount, start.Y * (1 - i / IntervalCount) + end.Y * i / IntervalCount));
00264                     }
00265                     else
00266                     {
00267                         pt = inv.ToDataCoordinates(new Point(start.X * 0.5 + end.X * 0.5, start.Y *
      0.5 + end.Y * 0.5));
00268                     }
00269                 }
00270                 else
00271                 {
00272                     for (int j = 0; j < pt.Length; j++)
00273                     {
00274                         pt[j] = StartPoint[j] + direction[j] / IntervalCount * i;
```

```
00275                     }
00276                 }
00277
00278
00279                 double[] prevPt = CoordinateSystem.GetAround(pt, invDir);
00280                 double[] nextPt = CoordinateSystem.GetAround(pt, normDir);
00281
00282                 Point point = CoordinateSystem.ToPlotCoordinates(pt);
00283                 Point prevPoint = CoordinateSystem.ToPlotCoordinates(prevPt);
00284                 Point nextPoint = CoordinateSystem.ToPlotCoordinates(nextPt);
00285
00286                 Point deriv = new Point(nextPoint.X - prevPoint.X, nextPoint.Y - prevPoint.Y);
00287                 double derivMod = deriv.Modulus();
00288                 deriv = new Point(deriv.X / derivMod, deriv.Y / derivMod);
00289                 Point perp = new Point(-deriv.Y, deriv.X);
00290
00291                 double sizeAbove = SizeAbove(i);
00292                 double sizeBelow = SizeBelow(i);
00293
00294                 if (sizeAbove + sizeBelow > 0)
00295                 {
00296                     tickPath.MoveTo(point.X - sizeAbove * perp.X, point.Y - sizeAbove * perp.Y);
00297                     tickPath.LineTo(point.X + sizeBelow * perp.X, point.Y + sizeBelow * perp.Y);
00298                 }
00299             }
00300
00301             target.StrokePath(tickPath, PresentationAttributes.Stroke,
      PresentationAttributes.LineWidth, PresentationAttributes.LineCap, PresentationAttributes.LineJoin,
      PresentationAttributes.LineDash, tag:  Tag);
00302         }
00303
00304     }
00305
00306 /// <summary>
00307 /// A plot element that draws equally spaced labels on an axis.
00308 /// </summary>
00309     public class ContinuousAxisLabels :  IPlotElement
00310     {
00311 /// <summary>
00312 /// The starting point of the axis, expressed in data space coordinates.
00313 /// </summary>
00314         public IReadOnlyList<double> StartPoint { get; set; }
00315
00316 /// <summary>
00317 /// The ending point of the axis, expressed in data space coordinates.
00318 /// </summary>
00319         public IReadOnlyList<double> EndPoint { get; set; }
00320
00321 /// <summary>
00322 /// The number of intervals between labels.  Note that the number of labels will be one greater than
      this.
00323 /// </summary>
00324         public double IntervalCount { get; set; } = 10;
00325
00326 /// <summary>
00327 /// The distance of the label anchor from the point on the axis.
00328 /// This should be set to a function accepting an <see langword="int"/> argument representing the
      index of the label, and
00329 /// return a <see langword="double"/> representing the distance of the label from the axis, in plot
      space coordinates.
00330 /// </summary>
00331         public Func<int, double> Position { get; set; } = _ => 10;
00332
00333 /// <summary>
00334 /// A function used to determine what text to draw at each label.  You should set this to a
00335 /// function accepting two parameters:  an <see cref="IReadOnlyList{T}"/> of <see
      langword="double"/>s,
00336 /// representing the coordinates of the point the label refers to, and an <see langword="int"/>
00337 /// representing the index of the label.  The function should return an <see cref="IEnumerable{T}" />
      of
00338 /// <see cref="FormattedText" /> objects, representing the text that will be drawn.
00339 /// </summary>
00340         public Func<IReadOnlyList<double>, int, IEnumerable<FormattedText» TextFormat { get; set; }
00341
00342 /// <summary>
00343 /// The orientation of the label with respect to the horizontal.  If this is <see langword="null"/>,
      the
00344 /// labels will be perpendicular to the axis.
00345 /// </summary>
00346         public double?  Rotation { get; set; } = null;
00347
00348         private IEnumerable<FormattedText> DefaultTextFormat(IReadOnlyList<double> point)
00349         {
00350             double[] direction = new double[StartPoint.Count];
00351             double directionMod = 0;
00352
00353             double maxDirection = 0;
```

```
00354             int maxDirectionIndex = 0;
00355
00356             for (int i = 0; i < direction.Length; i++)
00357             {
00358                 direction[i] = EndPoint[i] - StartPoint[i];
00359                 directionMod += direction[i] * direction[i];
00360
00361                 if (Math.Abs(direction[i]) > maxDirection)
00362                 {
00363                     maxDirection = Math.Abs(direction[i]);
00364                     maxDirectionIndex = i;
00365                 }
00366             }
00367
00368             directionMod = Math.Sqrt(directionMod);
00369
00370             if (maxDirection / directionMod >= 0.9)
00371             {
00372                 double range = Math.Abs((EndPoint[maxDirectionIndex] - StartPoint[maxDirectionIndex])
      / IntervalCount);
00373
00374                 string formatString;
00375
00376                 if (range >= 10)
00377                 {
00378                     formatString = "0";
00379                 }
00380                 else if (range >= 1)
00381                 {
00382                     formatString = "0.0";
00383                 }
00384                 else
00385                 {
00386                     formatString = "0." + new string('0', -(int)Math.Floor(Math.Log10(range)) + 1);
00387                 }
00388
00389                 if (PresentationAttributes.Font.FontFamily.IsStandardFamily)
00390                 {
00391                     int i = Array.IndexOf(FontFamily.StandardFamilies,
      PresentationAttributes.Font.FontFamily.FamilyName.Replace(" ", "-"));
00392
00393                     return FormattedText.Format(point[maxDirectionIndex].ToString(formatString,
      System.Globalization.CultureInfo.InvariantCulture), (FontFamily.StandardFontFamilies)i,
      PresentationAttributes.Font.FontSize);
00394                 }
00395                 else
00396                 {
00397                     return FormattedText.Format(point[maxDirectionIndex].ToString(formatString,
      System.Globalization.CultureInfo.InvariantCulture), PresentationAttributes.Font,
      PresentationAttributes.Font, PresentationAttributes.Font, PresentationAttributes.Font);
00398                 }
00399             }
00400             else
00401             {
00402                 double pointMod = 0;
00403
00404                 for (int i = 0; i < point.Count; i++)
00405                 {
00406                     pointMod += (point[i] - StartPoint[i]) * (point[i] - StartPoint[i]);
00407                 }
00408
00409                 if (PresentationAttributes.Font.FontFamily.IsStandardFamily)
00410                 {
00411                     int i = Array.IndexOf(FontFamily.StandardFamilies,
      PresentationAttributes.Font.FontFamily.FamilyName.Replace(" ", "-"));
00412
00413                     return FormattedText.Format((pointMod / directionMod).ToString("0%",
      System.Globalization.CultureInfo.InvariantCulture), (FontFamily.StandardFontFamilies)i,
      PresentationAttributes.Font.FontSize);
00414                 }
00415                 else
00416                 {
00417                     return FormattedText.Format((pointMod / directionMod).ToString("0%",
      System.Globalization.CultureInfo.InvariantCulture), PresentationAttributes.Font,
      PresentationAttributes.Font, PresentationAttributes.Font, PresentationAttributes.Font);
00418                 }
00419             }
00420         }
00421
00422 /// <summary>
00423 /// The baseline for the label anchor.
00424 /// </summary>
00425         public TextBaselines Baseline { get; set; } = TextBaselines.Middle;
00426
00427 /// <summary>
00428 /// The alignment for the label anchor.
00429 /// </summary>
```

```
00430         public TextAnchors Alignment { get; set; } = TextAnchors.Left;
00431
00432 /// <summary>
00433 /// The coordinate system used to transform the points from data space to plot space.
00434 /// </summary>
00435         public IContinuousCoordinateSystem CoordinateSystem { get; set; }
00436         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00437
00438 /// <summary>
00439 /// Presentation attributes determining the appearance of the labels.
00440 /// </summary>
00441         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes();
00442
00443 /// <summary>
00444 /// A tag to identify the labels in the plot.
00445 /// </summary>
00446         public string Tag { get; set; }
00447
00448 /// <summary>
00449 /// Create a new <see cref="ContinuousAxisLabels"/> instance.
00450 /// </summary>
00451 /// <param name="startPoint">The starting point of the axis, expressed in data space
      coordinates.</param>
00452 /// <param name="endPoint">The ending point of the axis, expressed in data space coordinates.</param>
00453 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00454         public ContinuousAxisLabels(IReadOnlyList<double> startPoint, IReadOnlyList<double> endPoint,
      IContinuousCoordinateSystem coordinateSystem)
00455         {
00456             this.StartPoint = startPoint;
00457             this.EndPoint = endPoint;
00458             this.CoordinateSystem = coordinateSystem;
00459             this.TextFormat = (p, i) => this.DefaultTextFormat(p);
00460         }
00461
00462 /// <inheritdoc/>
00463         public void Plot(Graphics target)
00464         {
00465             double[] direction = new double[StartPoint.Count];
00466             double directionMod = 0;
00467
00468             for (int i = 0; i < direction.Length; i++)
00469             {
00470                 direction[i] = EndPoint[i] - StartPoint[i];
00471                 directionMod += direction[i] * direction[i];
00472             }
00473
00474             directionMod = Math.Sqrt(directionMod);
00475
00476             double[] normDir = new double[StartPoint.Count];
00477             double[] invDir = new double[StartPoint.Count];
00478
00479             for (int i = 0; i < direction.Length; i++)
00480             {
00481                 normDir[i] = direction[i] / directionMod;
00482                 invDir[i] = -normDir[i];
00483             }
00484
00485             for (int i = 0; i <= IntervalCount; i++)
00486             {
00487                 double[] pt = new double[StartPoint.Count];
00488
00489                 if (CoordinateSystem is IContinuousInvertibleCoordinateSystem inv &&
      inv.IsDirectionStraight(direction))
00490                 {
00491                     Point start = inv.ToPlotCoordinates(StartPoint);
00492                     Point end = inv.ToPlotCoordinates(EndPoint);
00493
00494                     if (IntervalCount != 0)
00495                     {
00496                         pt = inv.ToDataCoordinates(new Point(start.X * (1 - i / IntervalCount) + end.X
      * i / IntervalCount, start.Y * (1 - i / IntervalCount) + end.Y * i / IntervalCount));
00497                     }
00498                     else
00499                     {
00500                         pt = inv.ToDataCoordinates(new Point(start.X * 0.5 + end.X * 0.5, start.Y *
      0.5 + end.Y * 0.5));
00501                     }
00502                 }
00503                 else
00504                 {
00505                     for (int j = 0; j < pt.Length; j++)
00506                     {
00507                         pt[j] = StartPoint[j] + direction[j] / IntervalCount * i;
00508                     }
00509                 }
```

```
00510
00511
00512                 double[] prevPt = CoordinateSystem.GetAround(pt, invDir);
00513                 double[] nextPt = CoordinateSystem.GetAround(pt, normDir);
00514
00515                 IEnumerable<FormattedText> text = TextFormat(pt, i);
00516
00517                 if (text != null)
00518                 {
00519                     Point point = CoordinateSystem.ToPlotCoordinates(pt);
00520                     Point prevPoint = CoordinateSystem.ToPlotCoordinates(prevPt);
00521                     Point nextPoint = CoordinateSystem.ToPlotCoordinates(nextPt);
00522
00523                     Point deriv = new Point(nextPoint.X - prevPoint.X, nextPoint.Y - prevPoint.Y);
00524                     double derivMod = deriv.Modulus();
00525                     deriv = new Point(deriv.X / derivMod, deriv.Y / derivMod);
00526                     Point perp = new Point(-deriv.Y, deriv.X);
00527
00528                     double position = Position(i);
00529
00530                     target.Save();
00531                     target.Translate(point.X + position * perp.X, point.Y + position * perp.Y);
00532
00533                     if (Rotation == null)
00534                     {
00535                         target.Rotate(Math.Atan2(perp.Y, perp.X));
00536                     }
00537                     else
00538                     {
00539                         target.Rotate(Rotation ??  0);
00540                     }
00541
00542
00543                     double x = Alignment == TextAnchors.Left ?  0 :  Alignment == TextAnchors.Right ?
      -text.Measure().Width :  -text.Measure().Width * 0.5;
00544
00545                     string fillTag = Tag;
00546                     string strokeTag = Tag;
00547
00548                     if (!string.IsNullOrEmpty(Tag) && target.UseUniqueTags)
00549                     {
00550                         fillTag = fillTag + "@" + i;
00551                         strokeTag = strokeTag + "@stroke" + i;
00552                     }
00553
00554                     target.FillText(x, 0, text, PresentationAttributes.Fill, Baseline, fillTag);
00555
00556                     if (PresentationAttributes.Stroke != null)
00557                     {
00558                         target.StrokeText(x, 0, text, PresentationAttributes.Stroke, Baseline,
      PresentationAttributes.LineWidth, PresentationAttributes.LineCap, PresentationAttributes.LineJoin,
      PresentationAttributes.LineDash, strokeTag);
00559                     }
00560
00561                     target.Restore();
00562                 }
00563             }
00564         }
00565     }
00566
00567 /// <summary>
00568 /// A plot element that draws a title for an axis.
00569 /// </summary>
00570     public class ContinuousAxisTitle :  IPlotElement
00571     {
00572 /// <summary>
00573 /// The starting point of the axis, expressed in data space coordinates.
00574 /// </summary>
00575         public IReadOnlyList<double> StartPoint { get; set; }
00576
00577 /// <summary>
00578 /// The ending point of the axis, expressed in data space coordinates.
00579 /// </summary>
00580         public IReadOnlyList<double> EndPoint { get; set; }
00581
00582 /// <summary>
00583 /// The distance between the title and the axis, in plot space coordinates.
00584 /// </summary>
00585         public double Position { get; set; } = 30;
00586
00587 /// <summary>
00588 /// The axis title to draw.
00589 /// </summary>
00590         public IEnumerable<FormattedText> Title { get; set; }
00591
00592 /// <summary>
00593 /// If the axis is not a straight line (e.g., because the coordinate system is not linear),
```

```
00594 /// if this is <see langword="true"/> the title will follow the shape of the axis.  If this
00595 /// is <see langword="false"/>, the title will always be drawn on a straight line.
00596 /// </summary>
00597         public bool FollowAxis { get; set; } = true;
00598 
00599 /// <summary>
00600 /// Orientation of the title with respect to the horizontal.  If this is <see langword="null"/>,
00601 /// the title is parallel to the axis.
00602 /// </summary>
00603         public double?  Rotation { get; set; } = null;
00604 
00605 /// <summary>
00606 /// The baseline for the title anchor.
00607 /// </summary>
00608         public TextBaselines Baseline { get; set; } = TextBaselines.Middle;
00609 
00610 /// <summary>
00611 /// The alignment for the title anchor.
00612 /// </summary>
00613         public TextAnchors Alignment { get; set; } = TextAnchors.Left;
00614 
00615 /// <summary>
00616 /// The coordinate system used to transform the points from data space to plot space.
00617 /// </summary>
00618         public IContinuousCoordinateSystem CoordinateSystem { get; set; }
00619         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00620 
00621 /// <summary>
00622 /// Presentation attributes determining the appearance of the title.
00623 /// </summary>
00624         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes() { Font = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 14), Stroke = null,
      Fill = Colours.Black };
00625 
00626 /// <summary>
00627 /// A tag to identify the labels in the plot.
00628 /// </summary>
00629         public string Tag { get; set; }
00630 
00631 /// <summary>
00632 /// Create a new <see cref="ContinuousAxisTitle"/> instance.
00633 /// </summary>
00634 /// <param name="title">The title to draw on the axis.</param>
00635 /// <param name="startPoint">The starting point of the axis, expressed in data space
      coordinates.</param>
00636 /// <param name="endPoint">The ending point of the axis, expressed in data space coordinates.</param>
00637 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00638         public ContinuousAxisTitle(IEnumerable<FormattedText> title, IReadOnlyList<double> startPoint,
      IReadOnlyList<double> endPoint, IContinuousCoordinateSystem coordinateSystem)
00639         {
00640             this.StartPoint = startPoint;
00641             this.EndPoint = endPoint;
00642             this.CoordinateSystem = coordinateSystem;
00643             this.Title = title;
00644         }
00645 
00646 /// <summary>
00647 /// Create a new <see cref="ContinuousAxisTitle"/> instance.
00648 /// </summary>
00649 /// <param name="title">The title to draw on the axis.</param>
00650 /// <param name="startPoint">The starting point of the axis, expressed in data space
      coordinates.</param>
00651 /// <param name="endPoint">The ending point of the axis, expressed in data space coordinates.</param>
00652 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00653 /// <param name="presentationAttributes">Presentation attributes determining the appearance of the
      title.</param>
00654         public ContinuousAxisTitle(string title, IReadOnlyList<double> startPoint,
      IReadOnlyList<double> endPoint, IContinuousCoordinateSystem coordinateSystem,
      PlotElementPresentationAttributes presentationAttributes = null)
00655         {
00656             this.StartPoint = startPoint;
00657             this.EndPoint = endPoint;
00658             this.CoordinateSystem = coordinateSystem;
00659 
00660             if (presentationAttributes != null)
00661             {
00662                 this.PresentationAttributes = presentationAttributes;
00663             }
00664 
00665             if (title != null)
00666             {
00667                 if (PresentationAttributes.Font.FontFamily.IsStandardFamily)
00668                 {
00669                     int i = Array.IndexOf(FontFamily.StandardFamilies,
```

```
PresentationAttributes.Font.FontFamily.FamilyName.Replace(" ", "-"));
00670
00671                     this.Title = FormattedText.Format(title, (FontFamily.StandardFontFamilies)i,
PresentationAttributes.Font.FontSize);
00672                 }
00673                 else
00674                 {
00675                     this.Title = FormattedText.Format(title, PresentationAttributes.Font,
PresentationAttributes.Font, PresentationAttributes.Font, PresentationAttributes.Font);
00676                 }
00677             }
00678             else
00679             {
00680                 Title = null;
00681             }
00682         }
00683
00684 /// <inheritdoc/>
00685         public void Plot(Graphics target)
00686         {
00687             if (Title != null)
00688             {
00689                 double[] direction = new double[StartPoint.Count];
00690                 double directionMod = 0;
00691
00692                 for (int i = 0; i < direction.Length; i++)
00693                 {
00694                     direction[i] = EndPoint[i] - StartPoint[i];
00695                     directionMod += direction[i] * direction[i];
00696                 }
00697
00698                 if (!FollowAxis || CoordinateSystem.IsLinear ||
CoordinateSystem.IsDirectionStraight(direction))
00699                 {
00700                     directionMod = Math.Sqrt(directionMod);
00701
00702                     double[] normDir = new double[StartPoint.Count];
00703                     double[] invDir = new double[StartPoint.Count];
00704
00705                     for (int i = 0; i < direction.Length; i++)
00706                     {
00707                         normDir[i] = direction[i] / directionMod;
00708                         invDir[i] = -normDir[i];
00709                     }
00710
00711                     double[] pt = new double[StartPoint.Count];
00712
00713                     if (CoordinateSystem is IContinuousInvertibleCoordinateSystem inv &&
inv.IsDirectionStraight(direction))
00714                     {
00715                         Point start = inv.ToPlotCoordinates(StartPoint);
00716                         Point end = inv.ToPlotCoordinates(EndPoint);
00717
00718                         pt = inv.ToDataCoordinates(new Point(start.X * 0.5 + end.X * 0.5, start.Y *
0.5 + end.Y * 0.5));
00719                     }
00720                     else
00721                     {
00722                         for (int j = 0; j < pt.Length; j++)
00723                         {
00724                             pt[j] = StartPoint[j] + direction[j] * 0.5;
00725                         }
00726                     }
00727
00728                     double[] prevPt = CoordinateSystem.GetAround(pt, invDir);
00729                     double[] nextPt = CoordinateSystem.GetAround(pt, normDir);
00730
00731                     Point point = CoordinateSystem.ToPlotCoordinates(pt);
00732                     Point prevPoint = CoordinateSystem.ToPlotCoordinates(prevPt);
00733                     Point nextPoint = CoordinateSystem.ToPlotCoordinates(nextPt);
00734
00735                     Point deriv = new Point(nextPoint.X - prevPoint.X, nextPoint.Y - prevPoint.Y);
00736                     double derivMod = deriv.Modulus();
00737                     deriv = new Point(deriv.X / derivMod, deriv.Y / derivMod);
00738                     Point perp = new Point(-deriv.Y, deriv.X);
00739
00740                     target.Save();
00741                     target.Translate(point.X + Position * perp.X, point.Y + Position * perp.Y);
00742
00743                     if (Rotation == null)
00744                     {
00745                         target.Rotate(Math.Atan2(perp.Y, perp.X) - Math.PI / 2);
00746                     }
00747                     else
00748                     {
00749                         target.Rotate((Rotation ??  0) - Math.PI / 2);
00750                     }
```

```
00751
00752
00753                   double x = Alignment == TextAnchors.Left ?  0 :  Alignment == TextAnchors.Right ?
    -Title.Measure().Width :  -Title.Measure().Width * 0.5;
00754
00755                   string fillTag = Tag;
00756                   string strokeTag = Tag;
00757
00758                   if (!string.IsNullOrEmpty(Tag) && target.UseUniqueTags)
00759                   {
00760                       strokeTag = strokeTag + "@stroke";
00761                   }
00762
00763                   target.FillText(x, 0, Title, PresentationAttributes.Fill, Baseline, fillTag);
00764
00765                   if (PresentationAttributes.Stroke != null)
00766                   {
00767                       target.StrokeText(x, 0, Title, PresentationAttributes.Stroke, Baseline,
    PresentationAttributes.LineWidth, PresentationAttributes.LineCap, PresentationAttributes.LineJoin,
    PresentationAttributes.LineDash, strokeTag);
00768                   }
00769
00770                   target.Restore();
00771               }
00772               else
00773               {
00774                   GraphicsPath path;
00775
00776                   path = new GraphicsPath();
00777
00778                   int count = 0;
00779
00780                   for (int i = 0; i < direction.Length; i++)
00781                   {
00782                       count = Math.Max(count, (int)Math.Ceiling(direction[i] /
    CoordinateSystem.Resolution[i]));
00783                   }
00784
00785                   double[] normDir = new double[StartPoint.Count];
00786                   double[] invDir = new double[StartPoint.Count];
00787
00788                   for (int i = 0; i < direction.Length; i++)
00789                   {
00790                       normDir[i] = direction[i] / directionMod;
00791                       invDir[i] = -normDir[i];
00792                   }
00793
00794                   for (int i = 0; i <= count; i++)
00795                   {
00796                       double[] pt = new double[StartPoint.Count];
00797
00798                       for (int j = 0; j < pt.Length; j++)
00799                       {
00800                           pt[j] = StartPoint[j] + direction[j] / count * i;
00801                       }
00802
00803                       double[] prevPt = CoordinateSystem.GetAround(pt, invDir);
00804                       double[] nextPt = CoordinateSystem.GetAround(pt, normDir);
00805
00806
00807                       Point point = CoordinateSystem.ToPlotCoordinates(pt);
00808                       Point prevPoint = CoordinateSystem.ToPlotCoordinates(prevPt);
00809                       Point nextPoint = CoordinateSystem.ToPlotCoordinates(nextPt);
00810
00811                       Point deriv = new Point(nextPoint.X - prevPoint.X, nextPoint.Y - prevPoint.Y);
00812                       double derivMod = deriv.Modulus();
00813                       deriv = new Point(deriv.X / derivMod, deriv.Y / derivMod);
00814                       Point perp = new Point(-deriv.Y, deriv.X);
00815
00816                       path.LineTo(point.X + perp.X * Position, point.Y + perp.Y * Position);
00817                   }
00818
00819                   string fillTag = Tag;
00820                   string strokeTag = Tag;
00821
00822                   if (!string.IsNullOrEmpty(Tag) && target.UseUniqueTags)
00823                   {
00824                       strokeTag = strokeTag + "@stroke";
00825                   }
00826
00827                   target.FillTextOnPath(path, Title.GetText(), PresentationAttributes.Font,
    PresentationAttributes.Fill, 0.5, Alignment, Baseline, fillTag);
00828
00829                   if (PresentationAttributes.Stroke != null)
00830                   {
00831                       target.StrokeTextOnPath(path, Title.GetText(), PresentationAttributes.Font,
    PresentationAttributes.Fill, 0.5, Alignment, Baseline, PresentationAttributes.LineWidth,
```

```
      PresentationAttributes.LineCap, PresentationAttributes.LineJoin, PresentationAttributes.LineDash,
      strokeTag);
00832                     }
00833                 }
00834             }
00835         }
00836     }
00837
00838 /// <summary>
00839 /// A plot element that draws a grid.
00840 /// </summary>
00841     public class Grid :  IPlotElement
00842     {
00843 /// <summary>
00844 /// The starting point for the first side of the grid.
00845 /// </summary>
00846         public IReadOnlyList<double> Side1Start { get; set; }
00847
00848 /// <summary>
00849 /// The ending point for the first side of the grid.
00850 /// </summary>
00851         public IReadOnlyList<double> Side1End { get; set; }
00852
00853 /// <summary>
00854 /// The starting point for the second side of the grid.
00855 /// </summary>
00856         public IReadOnlyList<double> Side2Start { get; set; }
00857
00858 /// <summary>
00859 /// The ending point for the second side of the grid.
00860 /// </summary>
00861         public IReadOnlyList<double> Side2End { get; set; }
00862
00863 /// <summary>
00864 /// The number of intervals between grid lines.  Note that the number of grid lines will be one
      greater than this.
00865 /// </summary>
00866         public int IntervalCount { get; set; } = 10;
00867
00868 /// <summary>
00869 /// The coordinate system used to transform the points from data space to plot space.
00870 /// </summary>
00871         public IContinuousCoordinateSystem CoordinateSystem { get; set; }
00872         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00873
00874 /// <summary>
00875 /// Presentation attributes determining the appearance of the grid.
00876 /// </summary>
00877         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(220, 220, 220)) };
00878
00879 /// <summary>
00880 /// A tag to identify the grid in the plot.
00881 /// </summary>
00882         public string Tag { get; set; }
00883
00884 /// <summary>
00885 /// Create a new <see cref="Grid"/> instance.
00886 /// </summary>
00887 /// <param name="side1Start">The starting point for the first side of the grid.</param>
00888 /// <param name="side1End">The ending point for the first side of the grid.</param>
00889 /// <param name="side2Start">The starting point for the second side of the grid.</param>
00890 /// <param name="side2End">The ending point for the second side of the grid.</param>
00891 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00892         public Grid(IReadOnlyList<double> side1Start, IReadOnlyList<double> side1End,
      IReadOnlyList<double> side2Start, IReadOnlyList<double> side2End, IContinuousCoordinateSystem
      coordinateSystem)
00893         {
00894             this.Side1Start = side1Start;
00895             this.Side1End = side1End;
00896             this.Side2Start = side2Start;
00897             this.Side2End = side2End;
00898             this.CoordinateSystem = coordinateSystem;
00899         }
00900
00901 /// <inheritdoc/>
00902         public void Plot(Graphics target)
00903         {
00904             IReadOnlyList<double> StartPoint1 = Side1Start;
00905             IReadOnlyList<double> EndPoint1 = Side1End;
00906             IReadOnlyList<double> StartPoint2 = Side2Start;
00907             IReadOnlyList<double> EndPoint2 = Side2End;
00908
00909             double[] direction1 = new double[StartPoint1.Count];
00910
00911             for (int i = 0; i < direction1.Length; i++)
```

```
00912             {
00913                 direction1[i] = EndPoint1[i] - StartPoint1[i];
00914             }
00915 
00916             double[] direction2 = new double[StartPoint2.Count];
00917 
00918             for (int i = 0; i < direction2.Length; i++)
00919             {
00920                 direction2[i] = EndPoint2[i] - StartPoint2[i];
00921             }
00922 
00923             GraphicsPath gridPath = new GraphicsPath();
00924 
00925             for (int i = 0; i <= IntervalCount; i++)
00926             {
00927                 double[] pt1 = new double[StartPoint1.Count];
00928                 double[] pt2 = new double[StartPoint2.Count];
00929 
00930                 if (CoordinateSystem is IContinuousInvertibleCoordinateSystem inv)
00931                 {
00932                     if (inv.IsDirectionStraight(direction1))
00933                     {
00934                         Point start = inv.ToPlotCoordinates(StartPoint1);
00935                         Point end = inv.ToPlotCoordinates(EndPoint1);
00936 
00937                         pt1 = inv.ToDataCoordinates(new Point(start.X * (1 - (double)i /
      IntervalCount) + end.X * i / IntervalCount, start.Y * (1 - (double)i / IntervalCount) + end.Y * i /
      IntervalCount));
00938                     }
00939                     else
00940                     {
00941                         for (int j = 0; j < pt1.Length; j++)
00942                         {
00943                             pt1[j] = StartPoint1[j] + direction1[j] / IntervalCount * i;
00944                         }
00945                     }
00946 
00947                     if (inv.IsDirectionStraight(direction2))
00948                     {
00949                         Point start = inv.ToPlotCoordinates(StartPoint2);
00950                         Point end = inv.ToPlotCoordinates(EndPoint2);
00951 
00952                         pt2 = inv.ToDataCoordinates(new Point(start.X * (1 - (double)i /
      IntervalCount) + end.X * i / IntervalCount, start.Y * (1 - (double)i / IntervalCount) + end.Y * i /
      IntervalCount));
00953                     }
00954                     else
00955                     {
00956                         for (int j = 0; j < pt1.Length; j++)
00957                         {
00958                             pt2[j] = StartPoint2[j] + direction2[j] / IntervalCount * i;
00959                         }
00960                     }
00961                 }
00962                 else
00963                 {
00964                     for (int j = 0; j < pt1.Length; j++)
00965                     {
00966                         pt1[j] = StartPoint1[j] + direction1[j] / IntervalCount * i;
00967                         pt2[j] = StartPoint2[j] + direction2[j] / IntervalCount * i;
00968                     }
00969                 }
00970 
00971 
00972 
00973 
00974                 Point point1 = CoordinateSystem.ToPlotCoordinates(pt1);
00975                 Point point2 = CoordinateSystem.ToPlotCoordinates(pt2);
00976 
00977 
00978                 gridPath.MoveTo(point1).LineTo(point2);
00979             }
00980 
00981             target.StrokePath(gridPath, PresentationAttributes.Stroke,
      PresentationAttributes.LineWidth, PresentationAttributes.LineCap, PresentationAttributes.LineJoin,
      PresentationAttributes.LineDash, tag:  Tag);
00982         }
00983     }
00984 }
```

8.37 Bars.cs

```
00001 /*
```

```
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018
00019 using System;
00020 using System.Collections.Generic;
00021 using System.Linq;
00022
00023 namespace VectSharp.Plots
00024 {
00025 /// <summary>
00026 /// A plot element that draws bars.
00027 /// </summary>
00028 /// <typeparam name="T">The type of data elements.</typeparam>
00029     public class Bars<T> :  IPlotElement
00030     {
00031         private double margin = 0;
00032
00033 /// <summary>
00034 /// The data points corresponding to the tips of the bars.
00035 /// </summary>
00036         public SortedSet<T> Data { get; set; }
00037
00038 /// <summary>
00039 /// A function that returns the bottom for each bar.  This function should accept
00040 /// a single parameter of type <typeparamref name="T"/> and return another <typeparamref name="T"/>
00041 /// object, representing the bottom of the bar in data space.
00042 /// </summary>
00043         public Func<T, T> GetBaseline { get; set; }
00044
00045 /// <summary>
00046 /// The margin between consecutive bars.  This should range between 0 and 1.
00047 /// </summary>
00048         public double Margin
00049         {
00050             get => margin;
00051             set
00052             {
00053                 if (value >= 0 && value <= 1)
00054                 {
00055                     margin = value;
00056                 }
00057                 else
00058                 {
00059                     throw new ArgumentOutOfRangeException(nameof(value), value, "The value for the
      margin must be within 0 and 1 (inclusive)!");
00060                 }
00061             }
00062         }
00063
00064 /// <summary>
00065 /// The coordinate system used to transform the points from data space to plot space.
00066 /// </summary>
00067         public ICoordinateSystem<T> CoordinateSystem { get; set; }
00068         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00069
00070 /// <summary>
00071 /// Presentation attributes for the bars.
00072 /// </summary>
00073         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes();
00074
00075 /// <summary>
00076 /// A tag to identify the bars in the plot.
00077 /// </summary>
00078         public string Tag { get; set; }
00079
00080 /// <summary>
00081 /// Create a new <see cref="Bars{T}"/> instance.
00082 /// </summary>
00083 /// <param name="data">The data points corresponding to the tips of the bars.</param>
00084 /// <param name="sorting">A comparer used to sort the bars.</param>
00085 /// <param name="getBaseline">A function that returns the bottom for each bar.  This function should
      accept
```

```
00086 /// a single parameter of type <typeparamref name="T"/> and return another <typeparamref name="T"/>
00087 /// object, representing the bottom of the bar in data space.</param>
00088 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00089         public Bars(IEnumerable<T> data, IComparer<T> sorting, Func<T, T> getBaseline,
      ICoordinateSystem<T> coordinateSystem)
00090         {
00091             this.Data = new SortedSet<T>(data, sorting);
00092             this.CoordinateSystem = coordinateSystem;
00093             this.GetBaseline = getBaseline;
00094         }
00095 
00096 /// <summary>
00097 /// Create a new <see cref="Bars{T}"/> instance.
00098 /// </summary>
00099 /// <param name="data">The data points corresponding to the tips of the bars.</param>
00100 /// <param name="sorting">A comparer used to sort the bars.</param>
00101 /// <param name="getBaseline">A function that returns the bottom for each bar.  This function should
      accept
00102 /// a single parameter of type <typeparamref name="T"/> and return another <typeparamref name="T"/>
00103 /// object, representing the bottom of the bar in data space.</param>
00104 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00105         public Bars(IEnumerable<T> data, Comparison<T> sorting, Func<T, T> getBaseline,
      ICoordinateSystem<T> coordinateSystem) :  this(data, Comparer<T>.Create(sorting), getBaseline,
      coordinateSystem) { }
00106 
00107 /// <summary>
00108 /// Create a new <see cref="Bars{T}"/> instance.
00109 /// </summary>
00110 /// <param name="data">The data points corresponding to the tips of the bars.  These should already be
      sorted.</param>
00111 /// <param name="getBaseline">A function that returns the bottom for each bar.  This function should
      accept
00112 /// a single parameter of type <typeparamref name="T"/> and return another <typeparamref name="T"/>
00113 /// object, representing the bottom of the bar in data space.</param>
00114 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00115         public Bars(IReadOnlyList<T> data, Func<T, T> getBaseline, ICoordinateSystem<T>
      coordinateSystem)
00116         {
00117             Dictionary<T, int> indices = new Dictionary<T, int>();
00118 
00119             for (int i = 0; i < data.Count; i++)
00120             {
00121                 indices[data[i]] = i;
00122             }
00123 
00124             this.Data = new SortedSet<T>(data, Comparer<T>.Create((x, y) => Math.Sign(indices[x] -
      indices[y])));
00125             this.CoordinateSystem = coordinateSystem;
00126             this.GetBaseline = getBaseline;
00127         }
00128 
00129         private static bool AnyNan(params Point[] points)
00130         {
00131             for (int i = 0; i < points.Length; i++)
00132             {
00133                 if (double.IsNaN(points[i].X) || double.IsNaN(points[i].Y))
00134                 {
00135                     return true;
00136                 }
00137             }
00138 
00139             return false;
00140         }
00141 
00142 /// <inheritdoc/>
00143         public void Plot(Graphics target)
00144         {
00145             List<(Point, Point)> bars = new List<(Point, Point)>();
00146 
00147             foreach (T data in Data)
00148             {
00149                 Point pt = CoordinateSystem.ToPlotCoordinates(data);
00150                 Point baseline = CoordinateSystem.ToPlotCoordinates(GetBaseline(data));
00151                 bars.Add((baseline, pt));
00152             }
00153 
00154             for (int i = 0; i < bars.Count; i++)
00155             {
00156                 GraphicsPath pth = new GraphicsPath();
00157 
00158                 if (i > 0 && i < bars.Count - 1)
00159                 {
00160                     Point prevBaselineMid = new Point(bars[i].Item1.X * (0.5 + Margin * 0.5) + bars[i
      - 1].Item1.X * (0.5 - Margin * 0.5), bars[i].Item1.Y * (0.5 + Margin * 0.5) + bars[i - 1].Item1.Y *
```

```
(0.5 - Margin * 0.5));
00161
00162                     Point perpDir = new Point(bars[i].Item2.Y - bars[i].Item1.Y, -(bars[i].Item2.X -
bars[i].Item1.X));
00163                     perpDir = perpDir.Normalize();
00164
00165                     double t = (bars[i].Item2.X - prevBaselineMid.X) * perpDir.X + (bars[i].Item2.Y -
prevBaselineMid.Y) * perpDir.Y;
00166                     Point prevTop = new Point(bars[i].Item2.X - perpDir.X * t, bars[i].Item2.Y -
perpDir.Y * t);
00167
00168                     Point nextBaselineMid = new Point(bars[i].Item1.X * (0.5 + Margin * 0.5) + bars[i
+ 1].Item1.X * (0.5 - Margin * 0.5), bars[i].Item1.Y * (0.5 + Margin * 0.5) + bars[i + 1].Item1.Y *
(0.5 - Margin * 0.5));
00169
00170                     double t2 = (bars[i].Item2.X - nextBaselineMid.X) * perpDir.X + (bars[i].Item2.Y -
nextBaselineMid.Y) * perpDir.Y;
00171                     Point nextTop = new Point(bars[i].Item2.X - perpDir.X * t2, bars[i].Item2.Y -
perpDir.Y * t2);
00172
00173                     if (!AnyNan(prevBaselineMid, prevTop, nextTop, nextBaselineMid))
00174                     {
00175
pth.MoveTo(prevBaselineMid).LineTo(prevTop).LineTo(nextTop).LineTo(nextBaselineMid).Close();
00176                     }
00177                 }
00178                 else if (i == 0)
00179                 {
00180                     Point perpDir = new Point(bars[i].Item2.Y - bars[i].Item1.Y, -(bars[i].Item2.X -
bars[i].Item1.X));
00181                     perpDir = perpDir.Normalize();
00182
00183                     Point nextBaselineMid = new Point(bars[i].Item1.X * (0.5 + Margin * 0.5) + bars[i
+ 1].Item1.X * (0.5 - Margin * 0.5), bars[i].Item1.Y * (0.5 + Margin * 0.5) + bars[i + 1].Item1.Y *
(0.5 - Margin * 0.5));
00184
00185                     double t2 = (bars[i].Item2.X - nextBaselineMid.X) * perpDir.X + (bars[i].Item2.Y -
nextBaselineMid.Y) * perpDir.Y;
00186                     Point nextTop = new Point(bars[i].Item2.X - perpDir.X * t2, bars[i].Item2.Y -
perpDir.Y * t2);
00187
00188
00189                     Point prevBaselineMid = new Point(2 * bars[i].Item1.X - nextBaselineMid.X, 2 *
bars[i].Item1.Y - nextBaselineMid.Y);
00190                     double t = -t2;
00191                     Point prevTop = new Point(bars[i].Item2.X - perpDir.X * t, bars[i].Item2.Y -
perpDir.Y * t);
00192
00193                     if (!AnyNan(prevBaselineMid, prevTop, nextTop, nextBaselineMid))
00194                     {
00195
pth.MoveTo(prevBaselineMid).LineTo(prevTop).LineTo(nextTop).LineTo(nextBaselineMid).Close();
00196                     }
00197                 }
00198                 else if (i == bars.Count - 1)
00199                 {
00200                     Point prevBaselineMid = new Point(bars[i].Item1.X * (0.5 + Margin * 0.5) + bars[i
- 1].Item1.X * (0.5 - Margin * 0.5), bars[i].Item1.Y * (0.5 + Margin * 0.5) + bars[i - 1].Item1.Y *
(0.5 - Margin * 0.5));
00201
00202                     Point perpDir = new Point(bars[i].Item2.Y - bars[i].Item1.Y, -(bars[i].Item2.X -
bars[i].Item1.X));
00203                     perpDir = perpDir.Normalize();
00204
00205                     double t = (bars[i].Item2.X - prevBaselineMid.X) * perpDir.X + (bars[i].Item2.Y -
prevBaselineMid.Y) * perpDir.Y;
00206                     Point prevTop = new Point(bars[i].Item2.X - perpDir.X * t, bars[i].Item2.Y -
perpDir.Y * t);
00207
00208                     Point nextBaselineMid = new Point(2 * bars[i].Item1.X - prevBaselineMid.X, 2 *
bars[i].Item1.Y - prevBaselineMid.Y);
00209
00210                     double t2 = -t;
00211                     Point nextTop = new Point(bars[i].Item2.X - perpDir.X * t2, bars[i].Item2.Y -
perpDir.Y * t2);
00212
00213                     if (!AnyNan(prevBaselineMid, prevTop, nextTop, nextBaselineMid))
00214                     {
00215
pth.MoveTo(prevBaselineMid).LineTo(prevTop).LineTo(nextTop).LineTo(nextBaselineMid).Close();
00216                     }
00217                 }
00218
00219                 string tag = Tag;
00220                 string strokeTag = Tag;
00221
00222                 if (!string.IsNullOrEmpty(Tag))
```

```
00223                 {
00224                     tag = tag + "@" + i.ToString();
00225                     strokeTag = tag + "@" + i.ToString() + "_stroke";
00226                 }
00227 
00228                 if (PresentationAttributes.Fill != null)
00229                 {
00230                     target.FillPath(pth, PresentationAttributes.Fill, tag);
00231                 }
00232 
00233                 if (PresentationAttributes.Stroke != null)
00234                 {
00235                     target.StrokePath(pth, PresentationAttributes.Stroke,
    PresentationAttributes.LineWidth, PresentationAttributes.LineCap, PresentationAttributes.LineJoin,
    PresentationAttributes.LineDash, strokeTag);
00236                 }
00237             }
00238         }
00239     }
00240 
00241 /// <summary>
00242 /// A plot element that draws bars for categorical data.
00243 /// </summary>
00244 /// <typeparam name="T">The type of the data categories.</typeparam>
00245     public class CategoricalBars<T> :  Bars<(T, double)>
00246     {
00247 /// <summary>
00248 /// Create a new <see cref="CategoricalBars{T}"/> instance.
00249 /// </summary>
00250 /// <param name="data">The data points corresponding to the tips of the bars.</param>
00251 /// <param name="sorting">A comparer used to sort the bars.</param>
00252 /// <param name="getBaseline">A function that returns the bottom for each bar.  This function should
    accept
00253 /// a single parameter of type <typeparamref name="T"/> and return another <typeparamref name="T"/>
00254 /// object, representing the bottom of the bar in data space.</param>
00255 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
    to plot space.</param>
00256         public CategoricalBars(IEnumerable<(T, double)> data, IComparer<(T, double)> sorting, Func<(T,
    double), (T, double)> getBaseline, ICoordinateSystem<(T, double)> coordinateSystem) :  base(data,
    sorting, getBaseline, coordinateSystem) { }
00257 
00258 /// <summary>
00259 /// Create a new <see cref="CategoricalBars{T}"/> instance.
00260 /// </summary>
00261 /// <param name="data">The data points corresponding to the tips of the bars.</param>
00262 /// <param name="sorting">A comparer used to sort the bars.</param>
00263 /// <param name="getBaseline">A function that returns the bottom for each bar.  This function should
    accept
00264 /// a single parameter of type <typeparamref name="T"/> and return another <typeparamref name="T"/>
00265 /// object, representing the bottom of the bar in data space.</param>
00266 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
    to plot space.</param>
00267         public CategoricalBars(IEnumerable<(T, double)> data, Comparison<(T, double)> sorting,
    Func<(T, double), (T, double)> getBaseline, ICoordinateSystem<(T, double)> coordinateSystem) :
    base(data, sorting, getBaseline, coordinateSystem) { }
00268 
00269 /// <summary>
00270 /// Create a new <see cref="Bars{T}"/> instance.  The baseline for each <c>(<typeparamref name="T"/>
    x, <see langword="double"/> y)</c> is determined automatically as <c>(x, 0)</c>.
00271 /// </summary>
00272 /// <param name="data">The data points corresponding to the tips of the bars.</param>
00273 /// <param name="sorting">A comparer used to sort the bars.</param>
00274 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
    to plot space.</param>
00275         public CategoricalBars(IEnumerable<(T, double)> data, Comparison<(T, double)> sorting,
    ICoordinateSystem<(T, double)> coordinateSystem) :  base(data, sorting, x => (x.Item1, 0),
    coordinateSystem) { }
00276 
00277 /// <summary>
00278 /// Create a new <see cref="Bars{T}"/> instance.  The baseline for each <c>(<typeparamref name="T"/>
    x, <see langword="double"/> y)</c> is determined automatically as <c>(x, 0)</c>.
00279 /// </summary>
00280 /// <param name="data">The data points corresponding to the tips of the bars.  These should already be
    sorted.</param>
00281 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
    to plot space.</param>
00282         public CategoricalBars(IReadOnlyList<(T, double)> data, ICoordinateSystem<(T, double)>
    coordinateSystem) :  base(data, x => (x.Item1, 0), coordinateSystem) { }
00283     }
00284 
00285 /// <summary>
00286 /// A plot element that draws bars for numerical data.
00287 /// </summary>
00288     public class Bars :  Bars<IReadOnlyList<double»
00289     {
00290 /// <summary>
00291 /// Create a new <see cref="Bars"/> instance.
```

```
00292 /// </summary>
00293 /// <param name="data">The data points corresponding to the tips of the bars.</param>
00294 /// <param name="sorting">A comparer used to sort the bars.</param>
00295 /// <param name="getBaseline">A function that returns the bottom for each bar.  This function should
      accept
00296 /// a single parameter (an <see cref="IReadOnlyList{T}"/> of <see langword="double"/>s), and return
      another
00297 /// object of the same type, representing the bottom of the bar in data space.</param>
00298 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00299         public Bars(IEnumerable<IReadOnlyList<double» data, IComparer<IReadOnlyList<double» sorting,
      Func<IReadOnlyList<double>, IReadOnlyList<double» getBaseline, ICoordinateSystem<IReadOnlyList<double»
      coordinateSystem) :  base(data, sorting, getBaseline, coordinateSystem) { }
00300
00301 /// <summary>
00302 /// Create a new <see cref="Bars"/> instance.
00303 /// </summary>
00304 /// <param name="data">The data points corresponding to the tips of the bars.</param>
00305 /// <param name="sorting">A comparer used to sort the bars.</param>
00306 /// <param name="getBaseline">A function that returns the bottom for each bar.  This function should
      accept
00307 /// a single parameter (an <see cref="IReadOnlyList{T}"/> of <see langword="double"/>s), and return
      another
00308 /// object of the same type, representing the bottom of the bar in data space.</param>
00309 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00310         public Bars(IEnumerable<IReadOnlyList<double» data, Comparison<IReadOnlyList<double» sorting,
      Func<IReadOnlyList<double>, IReadOnlyList<double» getBaseline, ICoordinateSystem<IReadOnlyList<double»
      coordinateSystem) :  base(data, sorting, getBaseline, coordinateSystem) { }
00311
00312 /// <summary>
00313 /// Create a new <see cref="Bars"/> instance.
00314 /// </summary>
00315 /// <param name="data">The data points corresponding to the tips of the bars.</param>
00316 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00317 /// <param name="vertical">If this is <see langword="true"/> (the default), the bars rise vertically
      above the X axis
00318 /// Otherwise, the bars grow horizontally from the Y axis.</param>
00319         public Bars(IEnumerable<IReadOnlyList<double» data, ICoordinateSystem<IReadOnlyList<double»
      coordinateSystem, bool vertical = true) :  base(data, vertical ?
      Comparer<IReadOnlyList<double».Create((x, y) => Math.Sign(x[0] - y[0])) :
      Comparer<IReadOnlyList<double».Create((x, y) => Math.Sign(x[1] - y[1])), vertical ?  new
      Func<IReadOnlyList<double>, IReadOnlyList<double»(x => { double[] tbr = x.ToArray(); tbr[1] = 0;
      return tbr; }) :  new Func<IReadOnlyList<double>, IReadOnlyList<double»(x => { double[] tbr =
      x.ToArray(); tbr[0] = 0; return tbr; }), coordinateSystem) { }
00320     }
00321
00322 /// <summary>
00323 /// A plot element that draws stacked bars.
00324 /// </summary>
00325     public class StackedBars :  IPlotElement
00326     {
00327         private double margin = 0;
00328
00329 /// <summary>
00330 /// If this is <see langword="true"/>, the bars rise vertically above the X axis
00331 /// Otherwise, the bars grow horizontally from the Y axis.
00332 /// </summary>
00333         public bool Vertical { get; set; } = true;
00334
00335 /// <summary>
00336 /// The data points corresponding to the tips of the bars.  For each bar stack, the data
00337 /// point contains an element determining the position of the bar on the X axis (if <see
      cref="Vertical"/>
00338 /// is <see langword="true"/>, or on the Y axis otherwise), and a set of elements determining the
00339 /// length of each segment in the bar stack.
00340 /// </summary>
00341         public SortedSet<IReadOnlyList<double» Data { get; set; }
00342
00343 /// <summary>
00344 /// A function that returns the bottom for each bar.  This function should accept
00345 /// a single parameter (an <see cref="IReadOnlyList{T}"/> of <see langword="double"/>s), and return
      another
00346 /// object of the same type, representing the bottom of the bar in data space.
00347 /// </summary>
00348         public Func<IReadOnlyList<double>, IReadOnlyList<double» GetBaseline { get; set; }
00349
00350 /// <summary>
00351 /// The margin between consecutive bars.  This should range between 0 and 1.
00352 /// </summary>
00353         public double Margin
00354         {
00355             get => margin;
00356             set
00357             {
00358                 if (value >= 0 && value <= 1)
```

```
00359                 {
00360                     margin = value;
00361                 }
00362                 else
00363                 {
00364                     throw new ArgumentOutOfRangeException(nameof(value), value, "The value for the
      margin must be within 0 and 1 (inclusive)!");
00365                 }
00366             }
00367         }
00368 
00369 /// <summary>
00370 /// The coordinate system used to transform the points from data space to plot space.
00371 /// </summary>
00372         public ICoordinateSystem<IReadOnlyList<double>> CoordinateSystem { get; set; }
00373         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00374 
00375 /// <summary>
00376 /// Presentation attributes for the bars.  An element from this collection is used for each segment in
00377 /// the stack; if there are more segments than elements in this collection, the presentation
      attributes
00378 /// are wrapped.
00379 /// </summary>
00380         public IReadOnlyList<PlotElementPresentationAttributes> PresentationAttributes { get; set; } =
      new PlotElementPresentationAttributes[] { new PlotElementPresentationAttributes() };
00381 
00382 /// <summary>
00383 /// A tag to identify the stacked bars in the plot.
00384 /// </summary>
00385         public string Tag { get; set; }
00386 
00387 /// <summary>
00388 /// Create a new <see cref="StackedBars"/> instance.
00389 /// </summary>
00390 /// <param name="data">The data points corresponding to the tips of the bars.  For each bar stack, the
      data
00391 /// point contains an element determining the position of the bar on the X axis (if <see
      cref="Vertical"/>
00392 /// is <see langword="true"/>, or on the Y axis otherwise), and a set of elements determining the
00393 /// length of each segment in the bar stack.</param>
00394 /// <param name="sorting">A comparer used to sort the bars.</param>
00395 /// <param name="getBaseline">A function that returns the bottom for each bar.  This function should
      accept
00396 /// a single parameter (an <see cref="IReadOnlyList{T}"/> of <see langword="double"/>s), and return
      another
00397 /// object of the same type, representing the bottom of the bar in data space.</param>
00398 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00399         public StackedBars(IEnumerable<IReadOnlyList<double» data, IComparer<IReadOnlyList<double»
      sorting, Func<IReadOnlyList<double>, IReadOnlyList<double» getBaseline,
      ICoordinateSystem<IReadOnlyList<double» coordinateSystem)
00400         {
00401             this.Data = new SortedSet<IReadOnlyList<double»(data, sorting);
00402             this.CoordinateSystem = coordinateSystem;
00403             this.GetBaseline = getBaseline;
00404         }
00405 
00406 /// <summary>
00407 /// Create a new <see cref="StackedBars"/> instance.
00408 /// </summary>
00409 /// <param name="data">The data points corresponding to the tips of the bars.  For each bar stack, the
      data
00410 /// point contains an element determining the position of the bar on the X axis (if <see
      cref="Vertical"/>
00411 /// is <see langword="true"/>, or on the Y axis otherwise), and a set of elements determining the
00412 /// length of each segment in the bar stack.</param>
00413 /// <param name="sorting">A comparer used to sort the bars.</param>
00414 /// <param name="getBaseline">A function that returns the bottom for each bar.  This function should
      accept
00415 /// a single parameter (an <see cref="IReadOnlyList{T}"/> of <see langword="double"/>s), and return
      another
00416 /// object of the same type, representing the bottom of the bar in data space.</param>
00417 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00418         public StackedBars(IEnumerable<IReadOnlyList<double» data, Comparison<IReadOnlyList<double»
      sorting, Func<IReadOnlyList<double>, IReadOnlyList<double» getBaseline,
      ICoordinateSystem<IReadOnlyList<double» coordinateSystem) :  this(data,
      Comparer<IReadOnlyList<double».Create(sorting), getBaseline, coordinateSystem) { }
00419 
00420 /// <summary>
00421 /// Create a new <see cref="StackedBars"/> instance.
00422 /// </summary>
00423 /// <param name="data">The data points corresponding to the tips of the bars.  For each bar stack, the
      data
00424 /// point contains an element determining the position of the bar on the X axis (if <paramref
      name="vertical"/>
00425 /// is <see langword="true"/>, or on the Y axis otherwise), and a set of elements determining the
```

```
00426 /// length of each segment in the bar stack.</param>
00427 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00428 /// <param name="vertical">If this is <see langword="true"/> (the default), the bars rise vertically
      above the X axis
00429 /// Otherwise, the bars grow horizontally from the Y axis.</param>
00430         public StackedBars(IEnumerable<IReadOnlyList<double» data,
      ICoordinateSystem<IReadOnlyList<double» coordinateSystem, bool vertical = true) :  this(data, vertical
      ?  Comparer<IReadOnlyList<double».Create((x, y) => Math.Sign(x[0] - y[0])) :
      Comparer<IReadOnlyList<double».Create((x, y) => Math.Sign(x[1] - y[1])), vertical ?  new
      Func<IReadOnlyList<double>, IReadOnlyList<double»(x => { double[] tbr = x.ToArray(); tbr[1] = 0;
      return tbr; }) :  new Func<IReadOnlyList<double>, IReadOnlyList<double»(x => { double[] tbr =
      x.ToArray(); tbr[0] = 0; return tbr; }), coordinateSystem) { Vertical = vertical; }
00431
00432 /// <inheritdoc/>
00433         public void Plot(Graphics target)
00434         {
00435             List<(Point, Point)> bars = new List<(Point, Point)>();
00436             List<double[]> ratios = new List<double[]>();
00437
00438             foreach (IReadOnlyList<double> data in Data)
00439             {
00440                 Point baseline = CoordinateSystem.ToPlotCoordinates(GetBaseline(data));
00441
00442                 double total = 0;
00443
00444                 double[] currRatios = new double[data.Count - 1];
00445
00446                 for (int i = 0; i < data.Count - 1; i++)
00447                 {
00448                     if (i == 0)
00449                     {
00450                         total += Vertical ?  data[1] :  data[0];
00451                     }
00452                     else
00453                     {
00454                         total += data[i + 1];
00455                     }
00456
00457                     Point currPt = CoordinateSystem.ToPlotCoordinates(Vertical ?  new double[] {
      data[0], total } :  new double[] { total, data[1] });
00458
00459                     currRatios[i] = new Point(currPt.X - baseline.X, currPt.Y - baseline.Y).Modulus();
00460                 }
00461
00462                 for (int i = 0; i < currRatios.Length; i++)
00463                 {
00464                     currRatios[i] /= currRatios[currRatios.Length - 1];
00465                 }
00466
00467                 Point pt = CoordinateSystem.ToPlotCoordinates(Vertical ?  new double[] { data[0],
      total } :  new double[] { total, data[1] });
00468
00469                 bars.Add((baseline, pt));
00470                 ratios.Add(currRatios);
00471             }
00472
00473             for (int i = 0; i < bars.Count; i++)
00474             {
00475                 List<GraphicsPath> paths = new List<GraphicsPath>();
00476
00477                 if (i > 0 && i < bars.Count - 1)
00478                 {
00479                     Point prevBaselineMid = new Point(bars[i].Item1.X * (0.5 + Margin * 0.5) + bars[i
      - 1].Item1.X * (0.5 - Margin * 0.5), bars[i].Item1.Y * (0.5 + Margin * 0.5) + bars[i - 1].Item1.Y *
      (0.5 - Margin * 0.5));
00480
00481                     Point perpDir = new Point(bars[i].Item2.Y - bars[i].Item1.Y, -(bars[i].Item2.X -
      bars[i].Item1.X));
00482                     perpDir = perpDir.Normalize();
00483
00484                     double t = (bars[i].Item2.X - prevBaselineMid.X) * perpDir.X + (bars[i].Item2.Y -
      prevBaselineMid.Y) * perpDir.Y;
00485                     Point prevTop = new Point(bars[i].Item2.X - perpDir.X * t, bars[i].Item2.Y -
      perpDir.Y * t);
00486
00487                     Point nextBaselineMid = new Point(bars[i].Item1.X * (0.5 + Margin * 0.5) + bars[i
      + 1].Item1.X * (0.5 - Margin * 0.5), bars[i].Item1.Y * (0.5 + Margin * 0.5) + bars[i + 1].Item1.Y *
      (0.5 - Margin * 0.5));
00488
00489                     double t2 = (bars[i].Item2.X - nextBaselineMid.X) * perpDir.X + (bars[i].Item2.Y -
      nextBaselineMid.Y) * perpDir.Y;
00490                     Point nextTop = new Point(bars[i].Item2.X - perpDir.X * t2, bars[i].Item2.Y -
      perpDir.Y * t2);
00491
00492                     Point lastBaseLeft = prevBaselineMid;
00493                     Point lastBaseRight = nextBaselineMid;
```

```
00494
00495                     for (int j = 0; j < ratios[i].Length; j++)
00496                     {
00497                         Point pTop = new Point(prevBaselineMid.X + (prevTop.X - prevBaselineMid.X) *
      ratios[i][j], prevBaselineMid.Y + (prevTop.Y - prevBaselineMid.Y) * ratios[i][j]);
00498                         Point nTop = new Point(nextBaselineMid.X + (nextTop.X - nextBaselineMid.X) *
      ratios[i][j], nextBaselineMid.Y + (nextTop.Y - nextBaselineMid.Y) * ratios[i][j]);
00499
00500                         paths.Add(new
      GraphicsPath().MoveTo(lastBaseLeft).LineTo(pTop).LineTo(nTop).LineTo(lastBaseRight).Close());
00501
00502                         lastBaseLeft = pTop;
00503                         lastBaseRight = nTop;
00504                     }
00505                 }
00506                 else if (i == 0 && bars.Count == 1)
00507                 {
00508                     Point perpDir = new Point(bars[i].Item2.Y - bars[i].Item1.Y, -(bars[i].Item2.X -
      bars[i].Item1.X));
00509                     perpDir = perpDir.Normalize();
00510
00511                     Point nextPoint = CoordinateSystem.ToPlotCoordinates(Vertical ?  new double[] {
      Data.First()[0] + 1, 0 } :  new double[] { 0, Data.First()[1] + 1 });
00512
00513                     Point nextBaselineMid = new Point(bars[i].Item1.X * (0.5 + Margin * 0.5) +
      nextPoint.X * (0.5 - Margin * 0.5), bars[i].Item1.Y * (0.5 + Margin * 0.5) + nextPoint.Y * (0.5 -
      Margin * 0.5));
00514
00515                     double t2 = (bars[i].Item2.X - nextBaselineMid.X) * perpDir.X + (bars[i].Item2.Y -
      nextBaselineMid.Y) * perpDir.Y;
00516                     Point nextTop = new Point(bars[i].Item2.X - perpDir.X * t2, bars[i].Item2.Y -
      perpDir.Y * t2);
00517
00518                     Point prevBaselineMid = new Point(2 * bars[i].Item1.X - nextBaselineMid.X, 2 *
      bars[i].Item1.Y - nextBaselineMid.Y);
00519                     double t = -t2;
00520                     Point prevTop = new Point(bars[i].Item2.X - perpDir.X * t, bars[i].Item2.Y -
      perpDir.Y * t);
00521
00522                     Point lastBaseLeft = prevBaselineMid;
00523                     Point lastBaseRight = nextBaselineMid;
00524
00525                     for (int j = 0; j < ratios[i].Length; j++)
00526                     {
00527                         Point pTop = new Point(prevBaselineMid.X + (prevTop.X - prevBaselineMid.X) *
      ratios[i][j], prevBaselineMid.Y + (prevTop.Y - prevBaselineMid.Y) * ratios[i][j]);
00528                         Point nTop = new Point(nextBaselineMid.X + (nextTop.X - nextBaselineMid.X) *
      ratios[i][j], nextBaselineMid.Y + (nextTop.Y - nextBaselineMid.Y) * ratios[i][j]);
00529
00530                         paths.Add(new
      GraphicsPath().MoveTo(lastBaseLeft).LineTo(pTop).LineTo(nTop).LineTo(lastBaseRight).Close());
00531
00532                         lastBaseLeft = pTop;
00533                         lastBaseRight = nTop;
00534                     }
00535                 }
00536                 else if (i == 0)
00537                 {
00538                     Point perpDir = new Point(bars[i].Item2.Y - bars[i].Item1.Y, -(bars[i].Item2.X -
      bars[i].Item1.X));
00539                     perpDir = perpDir.Normalize();
00540
00541                     Point nextBaselineMid = new Point(bars[i].Item1.X * (0.5 + Margin * 0.5) + bars[i
      + 1].Item1.X * (0.5 - Margin * 0.5), bars[i].Item1.Y * (0.5 + Margin * 0.5) + bars[i + 1].Item1.Y *
      (0.5 - Margin * 0.5));
00542
00543                     double t2 = (bars[i].Item2.X - nextBaselineMid.X) * perpDir.X + (bars[i].Item2.Y -
      nextBaselineMid.Y) * perpDir.Y;
00544                     Point nextTop = new Point(bars[i].Item2.X - perpDir.X * t2, bars[i].Item2.Y -
      perpDir.Y * t2);
00545
00546
00547                     Point prevBaselineMid = new Point(2 * bars[i].Item1.X - nextBaselineMid.X, 2 *
      bars[i].Item1.Y - nextBaselineMid.Y);
00548                     double t = -t2;
00549                     Point prevTop = new Point(bars[i].Item2.X - perpDir.X * t, bars[i].Item2.Y -
      perpDir.Y * t);
00550
00551                     Point lastBaseLeft = prevBaselineMid;
00552                     Point lastBaseRight = nextBaselineMid;
00553
00554                     for (int j = 0; j < ratios[i].Length; j++)
00555                     {
00556                         Point pTop = new Point(prevBaselineMid.X + (prevTop.X - prevBaselineMid.X) *
      ratios[i][j], prevBaselineMid.Y + (prevTop.Y - prevBaselineMid.Y) * ratios[i][j]);
00557                         Point nTop = new Point(nextBaselineMid.X + (nextTop.X - nextBaselineMid.X) *
      ratios[i][j], nextBaselineMid.Y + (nextTop.Y - nextBaselineMid.Y) * ratios[i][j]);
```

```
00558
00559                           paths.Add(new
      GraphicsPath().MoveTo(lastBaseLeft).LineTo(pTop).LineTo(nTop).LineTo(lastBaseRight).Close());
00560
00561                           lastBaseLeft = pTop;
00562                           lastBaseRight = nTop;
00563                       }
00564                   }
00565                   else if (i == bars.Count - 1)
00566                   {
00567                       Point prevBaselineMid = new Point(bars[i].Item1.X * (0.5 + Margin * 0.5) + bars[i
      - 1].Item1.X * (0.5 - Margin * 0.5), bars[i].Item1.Y * (0.5 + Margin * 0.5) + bars[i - 1].Item1.Y *
      (0.5 - Margin * 0.5));
00568
00569                       Point perpDir = new Point(bars[i].Item2.Y - bars[i].Item1.Y, -(bars[i].Item2.X -
      bars[i].Item1.X));
00570                       perpDir = perpDir.Normalize();
00571
00572                       double t = (bars[i].Item2.X - prevBaselineMid.X) * perpDir.X + (bars[i].Item2.Y -
      prevBaselineMid.Y) * perpDir.Y;
00573                       Point prevTop = new Point(bars[i].Item2.X - perpDir.X * t, bars[i].Item2.Y -
      perpDir.Y * t);
00574
00575                       Point nextBaselineMid = new Point(2 * bars[i].Item1.X - prevBaselineMid.X, 2 *
      bars[i].Item1.Y - prevBaselineMid.Y);
00576
00577                       double t2 = -t;
00578                       Point nextTop = new Point(bars[i].Item2.X - perpDir.X * t2, bars[i].Item2.Y -
      perpDir.Y * t2);
00579
00580                       Point lastBaseLeft = prevBaselineMid;
00581                       Point lastBaseRight = nextBaselineMid;
00582
00583                       for (int j = 0; j < ratios[i].Length; j++)
00584                       {
00585                           Point pTop = new Point(prevBaselineMid.X + (prevTop.X - prevBaselineMid.X) *
      ratios[i][j], prevBaselineMid.Y + (prevTop.Y - prevBaselineMid.Y) * ratios[i][j]);
00586                           Point nTop = new Point(nextBaselineMid.X + (nextTop.X - nextBaselineMid.X) *
      ratios[i][j], nextBaselineMid.Y + (nextTop.Y - nextBaselineMid.Y) * ratios[i][j]);
00587
00588                           paths.Add(new
      GraphicsPath().MoveTo(lastBaseLeft).LineTo(pTop).LineTo(nTop).LineTo(lastBaseRight).Close());
00589
00590                           lastBaseLeft = pTop;
00591                           lastBaseRight = nTop;
00592                       }
00593                   }
00594
00595                   for (int j = 0; j < paths.Count; j++)
00596                   {
00597                       string tag = Tag;
00598                       string strokeTag = Tag;
00599
00600                       if (!string.IsNullOrEmpty(Tag))
00601                       {
00602                           tag = tag + "@" + i.ToString() + "/" + j.ToString();
00603                           strokeTag = tag + "@" + i.ToString() + "/" + j.ToString() + "_stroke";
00604                       }
00605
00606                       int colourIndex = j % PresentationAttributes.Count;
00607
00608                       if (PresentationAttributes[colourIndex].Fill != null)
00609                       {
00610                           target.FillPath(paths[j], PresentationAttributes[colourIndex].Fill, tag);
00611                       }
00612
00613                       if (PresentationAttributes[colourIndex].Stroke != null)
00614                       {
00615                           target.StrokePath(paths[j], PresentationAttributes[colourIndex].Stroke,
      PresentationAttributes[colourIndex].LineWidth, PresentationAttributes[colourIndex].LineCap,
      PresentationAttributes[colourIndex].LineJoin, PresentationAttributes[colourIndex].LineDash,
      strokeTag);
00616                       }
00617                   }
00618               }
00619           }
00620       }
00621
00622 /// <summary>
00623 /// A plot element that draws clusters of bars.
00624 /// </summary>
00625       public class ClusteredBars :  IPlotElement
00626       {
00627           private double interClusterMargin = 0;
00628           private double intraClusterMargin = 0;
00629
00630 /// <summary>
```

```
00631 /// If this is <see langword="true"/>, the bars rise vertically above the X axis
00632 /// Otherwise, the bars grow horizontally from the Y axis.
00633 /// </summary>
00634         public bool Vertical { get; set; } = true;
00635 
00636 
00637 /// <summary>
00638 /// The data points corresponding to the tips of the bars.  For each bar cluster, the data
00639 /// point contains an element determining the position of the cluster on the X axis (if <see
      cref="Vertical"/>
00640 /// is <see langword="true"/>, or on the Y axis otherwise), and a set of elements determining the
00641 /// length of each bar in the cluster.
00642 /// </summary>
00643         public SortedSet<IReadOnlyList<double» Data { get; set; }
00644 
00645 /// <summary>
00646 /// A function that returns the bottom for each bar cluster.  This function should accept
00647 /// a single parameter (an <see cref="IReadOnlyList{T}"/> of <see langword="double"/>s), and return
      another
00648 /// object of the same type, representing the bottom of the cluster in data space.
00649 /// </summary>
00650         public Func<IReadOnlyList<double>, IReadOnlyList<double» GetBaseline { get; set; }
00651 
00652 /// <summary>
00653 /// The margin between consecutive bar clusters.
00654 /// </summary>
00655         public double InterClusterMargin
00656         {
00657             get => interClusterMargin;
00658             set
00659             {
00660                 if (value >= 0 && value <= 1)
00661                 {
00662                     interClusterMargin = value;
00663                 }
00664                 else
00665                 {
00666                     throw new ArgumentOutOfRangeException(nameof(value), value, "The value for the
      margin must be within 0 and 1 (inclusive)!");
00667                 }
00668             }
00669         }
00670 
00671 /// <summary>
00672 /// The margin between consecutive bars within a single cluster.
00673 /// </summary>
00674         public double IntraClusterMargin
00675         {
00676             get => intraClusterMargin;
00677             set
00678             {
00679                 if (value >= 0 && value <= 1)
00680                 {
00681                     intraClusterMargin = value;
00682                 }
00683                 else
00684                 {
00685                     throw new ArgumentOutOfRangeException(nameof(value), value, "The value for the
      margin must be within 0 and 1 (inclusive)!");
00686                 }
00687             }
00688         }
00689 
00690 /// <summary>
00691 /// The coordinate system used to transform the points from data space to plot space.
00692 /// </summary>
00693         public ICoordinateSystem<IReadOnlyList<double>> CoordinateSystem { get; set; }
00694         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00695 
00696 /// <summary>
00697 /// Presentation attributes for the bars.  An element from this collection is used for each bar in
00698 /// the cluster; if there are more bars than elements in this collection, the presentation attributes
00699 /// are wrapped.
00700 /// </summary>
00701         public IReadOnlyList<PlotElementPresentationAttributes> PresentationAttributes { get; set; } =
      new PlotElementPresentationAttributes[] { new PlotElementPresentationAttributes() };
00702 
00703 /// <summary>
00704 /// A tag to identify the clustered bars in the plot.
00705 /// </summary>
00706         public string Tag { get; set; }
00707 
00708 
00709 /// <summary>
00710 /// Create a new <see cref="ClusteredBars"/> instance.
00711 /// </summary>
00712 /// <param name="data">The data points corresponding to the tips of the bars.  For each bar cluster,
```

```
      the data
00713 /// point contains an element determining the position of the cluster on the X axis (if <see
      cref="Vertical"/>
00714 /// is <see langword="true"/>, or on the Y axis otherwise), and a set of elements determining the
00715 /// length of each bar in the cluster.</param>
00716 /// <param name="sorting">A comparer used to sort the bar clusters.</param>
00717 /// <param name="getBaseline">A function that returns the bottom for each bar cluster.  This function
      should accept
00718 /// a single parameter (an <see cref="IReadOnlyList{T}"/> of <see langword="double"/>s), and return
      another
00719 /// object of the same type, representing the bottom of the cluster in data space.</param>
00720 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00721         public ClusteredBars(IEnumerable<IReadOnlyList<double» data, IComparer<IReadOnlyList<double»
      sorting, Func<IReadOnlyList<double>, IReadOnlyList<double» getBaseline,
      ICoordinateSystem<IReadOnlyList<double» coordinateSystem)
00722         {
00723             this.Data = new SortedSet<IReadOnlyList<double»(data, sorting);
00724             this.CoordinateSystem = coordinateSystem;
00725             this.GetBaseline = getBaseline;
00726         }
00727
00728 /// <summary>
00729 /// Create a new <see cref="ClusteredBars"/> instance.
00730 /// </summary>
00731 /// <param name="data">The data points corresponding to the tips of the bars.  For each bar cluster,
      the data
00732 /// point contains an element determining the position of the cluster on the X axis (if <see
      cref="Vertical"/>
00733 /// is <see langword="true"/>, or on the Y axis otherwise), and a set of elements determining the
00734 /// length of each bar in the cluster.</param>
00735 /// <param name="sorting">A comparer used to sort the bar clusters.</param>
00736 /// <param name="getBaseline">A function that returns the bottom for each bar cluster.  This function
      should accept
00737 /// a single parameter (an <see cref="IReadOnlyList{T}"/> of <see langword="double"/>s), and return
      another
00738 /// object of the same type, representing the bottom of the cluster in data space.</param>
00739 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00740         public ClusteredBars(IEnumerable<IReadOnlyList<double» data, Comparison<IReadOnlyList<double»
      sorting, Func<IReadOnlyList<double>, IReadOnlyList<double» getBaseline,
      ICoordinateSystem<IReadOnlyList<double» coordinateSystem) :  this(data,
      Comparer<IReadOnlyList<double».Create(sorting), getBaseline, coordinateSystem) { }
00741
00742 /// <summary>
00743 /// Create a new <see cref="ClusteredBars"/> instance.
00744 /// </summary>
00745 /// <param name="data">The data points corresponding to the tips of the bars.  For each bar cluster,
      the data
00746 /// point contains an element determining the position of the cluster on the X axis (if <see
      cref="Vertical"/>
00747 /// is <see langword="true"/>, or on the Y axis otherwise), and a set of elements determining the
00748 /// length of each bar in the cluster.</param>
00749 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00750 /// <param name="vertical">If this is <see langword="true"/> (the default), the bars rise vertically
      above the X axis
00751 /// Otherwise, the bars grow horizontally from the Y axis.</param>
00752         public ClusteredBars(IEnumerable<IReadOnlyList<double» data,
      ICoordinateSystem<IReadOnlyList<double» coordinateSystem, bool vertical = true) :  this(data, vertical
      ?  Comparer<IReadOnlyList<double».Create((x, y) => Math.Sign(x[0] - y[0])) :
      Comparer<IReadOnlyList<double».Create((x, y) => Math.Sign(x[1] - y[1])), vertical ?  new
      Func<IReadOnlyList<double>, IReadOnlyList<double»(x => { double[] tbr = x.ToArray(); tbr[1] = 0;
      return tbr; }) :  new Func<IReadOnlyList<double>, IReadOnlyList<double»(x => { double[] tbr =
      x.ToArray(); tbr[0] = 0; return tbr; }), coordinateSystem) { Vertical = vertical; }
00753
00754 /// <inheritdoc/>
00755         public void Plot(Graphics target)
00756         {
00757             List<(Point, Point[])> bars = new List<(Point, Point[])>();
00758
00759             foreach (IReadOnlyList<double> data in Data)
00760             {
00761                 Point baseline = CoordinateSystem.ToPlotCoordinates(GetBaseline(data));
00762
00763                 Point[] points = new Point[data.Count - 1];
00764
00765                 for (int i = 0; i < data.Count - 1; i++)
00766                 {
00767                     if (i == 0)
00768                     {
00769                         points[i] = CoordinateSystem.ToPlotCoordinates(data);
00770                     }
00771                     else
00772                     {
00773                         points[i] = CoordinateSystem.ToPlotCoordinates(Vertical ?  new double[] {
      data[0], data[i + 1] } :  new double[] { data[i + 1], data[1] });
```

```
00774                     }
00775                 }
00776
00777                 bars.Add((baseline, points));
00778             }
00779
00780             for (int i = 0; i < bars.Count; i++)
00781             {
00782                 List<GraphicsPath> paths = new List<GraphicsPath>();
00783
00784                 if (i > 0 && i < bars.Count - 1)
00785                 {
00786                     Point prevBaselineMid = new Point(bars[i].Item1.X * (0.5 + InterClusterMargin *
      0.5) + bars[i - 1].Item1.X * (0.5 - InterClusterMargin * 0.5), bars[i].Item1.Y * (0.5 +
      InterClusterMargin * 0.5) + bars[i - 1].Item1.Y * (0.5 - InterClusterMargin * 0.5));
00787                     Point nextBaselineMid = new Point(bars[i].Item1.X * (0.5 + InterClusterMargin *
      0.5) + bars[i + 1].Item1.X * (0.5 - InterClusterMargin * 0.5), bars[i].Item1.Y * (0.5 +
      InterClusterMargin * 0.5) + bars[i + 1].Item1.Y * (0.5 - InterClusterMargin * 0.5));
00788
00789                     for (int j = 0; j < bars[i].Item2.Length; j++)
00790                     {
00791                         Point perpDir = new Point(bars[i].Item2[j].Y - bars[i].Item1.Y,
      -(bars[i].Item2[j].X - bars[i].Item1.X));
00792                         perpDir = perpDir.Normalize();
00793
00794                         double t = (bars[i].Item2[j].X - prevBaselineMid.X) * perpDir.X +
      (bars[i].Item2[j].Y - prevBaselineMid.Y) * perpDir.Y;
00795                         Point prevTop = new Point(bars[i].Item2[j].X - perpDir.X * t,
      bars[i].Item2[j].Y - perpDir.Y * t);
00796
00797                         double t2 = (bars[i].Item2[j].X - nextBaselineMid.X) * perpDir.X +
      (bars[i].Item2[j].Y - nextBaselineMid.Y) * perpDir.Y;
00798                         Point nextTop = new Point(bars[i].Item2[j].X - perpDir.X * t2,
      bars[i].Item2[j].Y - perpDir.Y * t2);
00799
00800                         double start = (double)(j + IntraClusterMargin * 0.5) / bars[i].Item2.Length;
00801                         double end = (double)(j + 1 - IntraClusterMargin * 0.5) /
      bars[i].Item2.Length;
00802
00803                         paths.Add(new GraphicsPath().MoveTo(new Point(prevBaselineMid.X +
      (nextBaselineMid.X - prevBaselineMid.X) * start, prevBaselineMid.Y + (nextBaselineMid.Y -
      prevBaselineMid.Y) * start))
00804                             .LineTo(new Point(prevTop.X + (nextTop.X - prevTop.X) * start, prevTop.Y +
      (nextTop.Y - prevTop.Y) * start))
00805                             .LineTo(new Point(prevTop.X + (nextTop.X - prevTop.X) * end, prevTop.Y +
      (nextTop.Y - prevTop.Y) * end))
00806                             .LineTo(new Point(prevBaselineMid.X + (nextBaselineMid.X -
      prevBaselineMid.X) * end, prevBaselineMid.Y + (nextBaselineMid.Y - prevBaselineMid.Y) *
      end)).Close());
00807                     }
00808                 }
00809                 else if (i == 0 && i == bars.Count - 1)
00810                 {
00811                     Point nextPoint = CoordinateSystem.ToPlotCoordinates(Vertical ?  new double[] {
      Data.First()[0] + 1, 0 } :  new double[] { 0, Data.First()[1] + 1 });
00812
00813                     Point nextBaselineMid = new Point(bars[i].Item1.X * (0.5 + InterClusterMargin *
      0.5) + nextPoint.X * (0.5 - InterClusterMargin * 0.5), bars[i].Item1.Y * (0.5 + InterClusterMargin *
      0.5) + nextPoint.Y * (0.5 - InterClusterMargin * 0.5));
00814
00815                     for (int j = 0; j < bars[i].Item2.Length; j++)
00816                     {
00817                         Point perpDir = new Point(bars[i].Item2[j].Y - bars[i].Item1.Y,
      -(bars[i].Item2[j].X - bars[i].Item1.X));
00818                         perpDir = perpDir.Normalize();
00819
00820                         double t2 = (bars[i].Item2[j].X - nextBaselineMid.X) * perpDir.X +
      (bars[i].Item2[j].Y - nextBaselineMid.Y) * perpDir.Y;
00821                         Point nextTop = new Point(bars[i].Item2[j].X - perpDir.X * t2,
      bars[i].Item2[j].Y - perpDir.Y * t2);
00822
00823                         Point prevBaselineMid = new Point(2 * bars[i].Item1.X - nextBaselineMid.X, 2 *
      bars[i].Item1.Y - nextBaselineMid.Y);
00824                         double t = -t2;
00825                         Point prevTop = new Point(bars[i].Item2[j].X - perpDir.X * t,
      bars[i].Item2[j].Y - perpDir.Y * t);
00826
00827                         double start = (double)(j + IntraClusterMargin * 0.5) / bars[i].Item2.Length;
00828                         double end = (double)(j + 1 - IntraClusterMargin * 0.5) /
      bars[i].Item2.Length;
00829
00830                         paths.Add(new GraphicsPath().MoveTo(new Point(prevBaselineMid.X +
      (nextBaselineMid.X - prevBaselineMid.X) * start, prevBaselineMid.Y + (nextBaselineMid.Y -
      prevBaselineMid.Y) * start))
00831                             .LineTo(new Point(prevTop.X + (nextTop.X - prevTop.X) * start, prevTop.Y +
      (nextTop.Y - prevTop.Y) * start))
00832                             .LineTo(new Point(prevTop.X + (nextTop.X - prevTop.X) * end, prevTop.Y +
```

```
(nextTop.Y - prevTop.Y) * end))
00833                           .LineTo(new Point(prevBaselineMid.X + (nextBaselineMid.X -
prevBaselineMid.X) * end, prevBaselineMid.Y + (nextBaselineMid.Y - prevBaselineMid.Y) *
end)).Close());
00834                   }
00835               }
00836               else if (i == 0)
00837               {
00838                   Point nextBaselineMid = new Point(bars[i].Item1.X * (0.5 + InterClusterMargin *
0.5) + bars[i + 1].Item1.X * (0.5 - InterClusterMargin * 0.5), bars[i].Item1.Y * (0.5 +
InterClusterMargin * 0.5) + bars[i + 1].Item1.Y * (0.5 - InterClusterMargin * 0.5));
00839
00840                   for (int j = 0; j < bars[i].Item2.Length; j++)
00841                   {
00842                       Point perpDir = new Point(bars[i].Item2[j].Y - bars[i].Item1.Y,
-(bars[i].Item2[j].X - bars[i].Item1.X));
00843                       perpDir = perpDir.Normalize();
00844
00845                       double t2 = (bars[i].Item2[j].X - nextBaselineMid.X) * perpDir.X +
(bars[i].Item2[j].Y - nextBaselineMid.Y) * perpDir.Y;
00846                       Point nextTop = new Point(bars[i].Item2[j].X - perpDir.X * t2,
bars[i].Item2[j].Y - perpDir.Y * t2);
00847
00848                       Point prevBaselineMid = new Point(2 * bars[i].Item1.X - nextBaselineMid.X, 2 *
bars[i].Item1.Y - nextBaselineMid.Y);
00849                       double t = -t2;
00850                       Point prevTop = new Point(bars[i].Item2[j].X - perpDir.X * t,
bars[i].Item2[j].Y - perpDir.Y * t);
00851
00852                       double start = (double)(j + IntraClusterMargin * 0.5) / bars[i].Item2.Length;
00853                       double end = (double)(j + 1 - IntraClusterMargin * 0.5) /
bars[i].Item2.Length;
00854
00855                       paths.Add(new GraphicsPath().MoveTo(new Point(prevBaselineMid.X +
(nextBaselineMid.X - prevBaselineMid.X) * start, prevBaselineMid.Y + (nextBaselineMid.Y -
prevBaselineMid.Y) * start))
00856                           .LineTo(new Point(prevTop.X + (nextTop.X - prevTop.X) * start, prevTop.Y +
(nextTop.Y - prevTop.Y) * start))
00857                           .LineTo(new Point(prevTop.X + (nextTop.X - prevTop.X) * end, prevTop.Y +
(nextTop.Y - prevTop.Y) * end))
00858                           .LineTo(new Point(prevBaselineMid.X + (nextBaselineMid.X -
prevBaselineMid.X) * end, prevBaselineMid.Y + (nextBaselineMid.Y - prevBaselineMid.Y) *
end)).Close());
00859                   }
00860               }
00861               else if (i == bars.Count - 1)
00862               {
00863                   Point prevBaselineMid = new Point(bars[i].Item1.X * (0.5 + InterClusterMargin *
0.5) + bars[i - 1].Item1.X * (0.5 - InterClusterMargin * 0.5), bars[i].Item1.Y * (0.5 +
InterClusterMargin * 0.5) + bars[i - 1].Item1.Y * (0.5 - InterClusterMargin * 0.5));
00864
00865                   for (int j = 0; j < bars[i].Item2.Length; j++)
00866                   {
00867                       Point perpDir = new Point(bars[i].Item2[j].Y - bars[i].Item1.Y,
-(bars[i].Item2[j].X - bars[i].Item1.X));
00868                       perpDir = perpDir.Normalize();
00869
00870                       double t = (bars[i].Item2[j].X - prevBaselineMid.X) * perpDir.X +
(bars[i].Item2[j].Y - prevBaselineMid.Y) * perpDir.Y;
00871                       Point prevTop = new Point(bars[i].Item2[j].X - perpDir.X * t,
bars[i].Item2[j].Y - perpDir.Y * t);
00872
00873                       Point nextBaselineMid = new Point(2 * bars[i].Item1.X - prevBaselineMid.X, 2 *
bars[i].Item1.Y - prevBaselineMid.Y);
00874
00875                       double t2 = -t;
00876                       Point nextTop = new Point(bars[i].Item2[j].X - perpDir.X * t2,
bars[i].Item2[j].Y - perpDir.Y * t2);
00877
00878                       double start = (double)(j + IntraClusterMargin * 0.5) / bars[i].Item2.Length;
00879                       double end = (double)(j + 1 - IntraClusterMargin * 0.5) /
bars[i].Item2.Length;
00880
00881                       paths.Add(new GraphicsPath().MoveTo(new Point(prevBaselineMid.X +
(nextBaselineMid.X - prevBaselineMid.X) * start, prevBaselineMid.Y + (nextBaselineMid.Y -
prevBaselineMid.Y) * start))
00882                           .LineTo(new Point(prevTop.X + (nextTop.X - prevTop.X) * start, prevTop.Y +
(nextTop.Y - prevTop.Y) * start))
00883                           .LineTo(new Point(prevTop.X + (nextTop.X - prevTop.X) * end, prevTop.Y +
(nextTop.Y - prevTop.Y) * end))
00884                           .LineTo(new Point(prevBaselineMid.X + (nextBaselineMid.X -
prevBaselineMid.X) * end, prevBaselineMid.Y + (nextBaselineMid.Y - prevBaselineMid.Y) *
end)).Close());
00885                   }
00886               }
00887
00888               for (int j = 0; j < paths.Count; j++)
```

```
00889                 {
00890                     string tag = Tag;
00891                     string strokeTag = Tag;
00892 
00893                     if (!string.IsNullOrEmpty(Tag))
00894                     {
00895                         tag = tag + "@" + i.ToString() + "/" + j.ToString();
00896                         strokeTag = tag + "@" + i.ToString() + "/" + j.ToString() + "_stroke";
00897                     }
00898 
00899                     int colourIndex = j % PresentationAttributes.Count;
00900 
00901                     if (PresentationAttributes[colourIndex].Fill != null)
00902                     {
00903                         target.FillPath(paths[j], PresentationAttributes[colourIndex].Fill, tag);
00904                     }
00905 
00906                     if (PresentationAttributes[colourIndex].Stroke != null)
00907                     {
00908                         target.StrokePath(paths[j], PresentationAttributes[colourIndex].Stroke,
      PresentationAttributes[colourIndex].LineWidth, PresentationAttributes[colourIndex].LineCap,
      PresentationAttributes[colourIndex].LineJoin, PresentationAttributes[colourIndex].LineDash,
      strokeTag);
00909                     }
00910                 }
00911             }
00912         }
00913     }
00914 }
```

8.38 BoxPlot.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
00019 using System.Collections.Generic;
00020 
00021 namespace VectSharp.Plots
00022 {
00023 /// <summary>
00024 /// A plot element that draws a box plot.
00025 /// </summary>
00026     public class BoxPlot :  IPlotElement
00027     {
00028 /// <summary>
00029 /// The position of the centre (e.g., median or mean) of the box plot, in data space coordinates.
00030 /// </summary>
00031         public IReadOnlyList<double> Position { get; set; }
00032 
00033 /// <summary>
00034 /// The direction along which the box plot is drawn, in data space coordinates.
00035 /// </summary>
00036         public IReadOnlyList<double> Direction { get; set; }
00037 
00038 /// <summary>
00039 /// The distance between the centre of the box plot and the first whisker, expressed as a multiple
00040 /// of <see cref="Direction"/>.
00041 /// </summary>
00042         public double Whisker1 { get; set; }
00043 
00044 /// <summary>
00045 /// The distance between the centre of the box plot and the first side of the box, expressed as a
      multiple
00046 /// of <see cref="Direction"/>.
00047 /// </summary>
00048         public double Box1 { get; set; }
00049 
00050 /// <summary>
```

```
00051 /// The distance between the centre of the box plot and the second side of the box, expressed as a
      multiple
00052 /// of <see cref="Direction"/>.
00053 /// </summary>
00054         public double Box2 { get; set; }
00055
00056 /// <summary>
00057 /// The distance between the centre of the box plot and the second whisker, expressed as a multiple
00058 /// of <see cref="Direction"/>.
00059 /// </summary>
00060         public double Whisker2 { get; set; }
00061
00062 /// <summary>
00063 /// The width of the box plot, in data space coordinates.
00064 /// </summary>
00065         public double Width { get; set; } = 10;
00066
00067 /// <summary>
00068 /// The width of the whiskers, expressed as a multiple of the <see cref="Width"/> of the box.
00069 /// </summary>
00070         public double WhiskerWidth { get; set; } = 0.5;
00071
00072 /// <summary>
00073 /// The width of the notch, expressed as a multiple of the <see cref="Width"/> of the box.
00074 /// </summary>
00075         public double NotchWidth { get; set; } = 0.5;
00076
00077 /// <summary>
00078 /// The distance between the centre of the box plot and the end of the notch, expressed as a multiple
00079 /// of <see cref="Direction"/>.
00080 /// </summary>
00081         public double NotchSize { get; set; } = 0;
00082
00083 /// <summary>
00084 /// Presentation attributes for the box.
00085 /// </summary>
00086         public PlotElementPresentationAttributes BoxPresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes() { Fill = Colours.White };
00087
00088 /// <summary>
00089 /// Presentation attributes for the whiskers.
00090 /// </summary>
00091         public PlotElementPresentationAttributes WhiskersPresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes() { Fill = null };
00092
00093 /// <summary>
00094 /// Presentation attributes for the symbol drawn at the centre of the box plot.
00095 /// </summary>
00096         public PlotElementPresentationAttributes CentreSymbolPresentationAttributes { get; set; } =
      new PlotElementPresentationAttributes() { Fill = null, Stroke = null };
00097
00098 /// <summary>
00099 /// Symbol drawn at the centre of the box plot.
00100 /// </summary>
00101         public IDataPointElement CentreSymbol { get; set; } = new PathDataPointElement();
00102
00103 /// <summary>
00104 /// The coordinate system used to transform the points from data space to plot space.
00105 /// </summary>
00106         public ICoordinateSystem<IReadOnlyList<double>> CoordinateSystem { get; set; }
00107
00108 /// <summary>
00109 /// A tag to identify the box in the plot.
00110 /// </summary>
00111         public string Tag { get; set; }
00112
00113         ICoordinateSystem IPlotElement.CoordinateSystem => this.CoordinateSystem;
00114
00115 /// <summary>
00116 /// Create a new <see cref="BoxPlot"/> instance.
00117 /// </summary>
00118 /// <param name="position">The position of the centre (e.g., median or mean) of the box plot, in data
      space coordinates.</param>
00119 /// <param name="direction">The direction along which the box plot is drawn, in data space
      coordinates.</param>
00120 /// <param name="whisker1">The distance between the centre of the box plot and the first whisker,
      expressed as a multiple
00121 /// of <paramref name="direction"/>.</param>
00122 /// <param name="box1">The distance between the centre of the box plot and the first side of the box,
      expressed as a multiple
00123 /// of <paramref name="direction"/>.</param>
00124 /// <param name="box2">The distance between the centre of the box plot and the second side of the box,
      expressed as a multiple
00125 /// of <paramref name="direction"/>.</param>
00126 /// <param name="whisker2">The distance between the centre of the box plot and the second whisker,
      expressed as a multiple
00127 /// of <paramref name="direction"/>.</param>
```

```
00128 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00129         public BoxPlot(IReadOnlyList<double> position, IReadOnlyList<double> direction, double
      whisker1, double box1, double box2, double whisker2, ICoordinateSystem<IReadOnlyList<double>>
      coordinateSystem)
00130         {
00131             this.Position = position;
00132             this.Direction = direction;
00133             this.Whisker1 = whisker1;
00134             this.Box1 = box1;
00135             this.Box2 = box2;
00136             this.Whisker2 = whisker2;
00137             this.CoordinateSystem = coordinateSystem;
00138         }
00139 
00140 /// <inheritdoc/>
00141         public void Plot(Graphics target)
00142         {
00143             double[] perpDirection = new double[] { -Direction[1], Direction[0] };
00144 
00145             Point whisker1 = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] +
      this.Direction[0] * this.Whisker1, this.Position[1] + this.Direction[1] * this.Whisker1 });
00146             Point whisker2 = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] +
      this.Direction[0] * this.Whisker2, this.Position[1] + this.Direction[1] * this.Whisker2 });
00147 
00148             Point whisker1Left = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] +
      this.Direction[0] * this.Whisker1 + perpDirection[0] * this.Width * this.WhiskerWidth,
      this.Position[1] + this.Direction[1] * this.Whisker1 + perpDirection[1] * this.Width *
      this.WhiskerWidth });
00149             Point box1Left = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] +
      this.Direction[0] * this.Box1 + perpDirection[0] * this.Width, this.Position[1] + this.Direction[1] *
      this.Box1 + perpDirection[1] * this.Width });
00150             Point box2Left = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] +
      this.Direction[0] * this.Box2 + perpDirection[0] * this.Width, this.Position[1] + this.Direction[1] *
      this.Box2 + perpDirection[1] * this.Width });
00151             Point whisker2Left = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] +
      this.Direction[0] * this.Whisker2 + perpDirection[0] * this.Width * this.WhiskerWidth,
      this.Position[1] + this.Direction[1] * this.Whisker2 + perpDirection[1] * this.Width *
      this.WhiskerWidth });
00152 
00153             Point whisker1Right = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] +
      this.Direction[0] * this.Whisker1 - perpDirection[0] * this.Width * this.WhiskerWidth,
      this.Position[1] + this.Direction[1] * this.Whisker1 - perpDirection[1] * this.Width *
      this.WhiskerWidth });
00154             Point box1Right = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] +
      this.Direction[0] * this.Box1 - perpDirection[0] * this.Width, this.Position[1] + this.Direction[1] *
      this.Box1 - perpDirection[1] * this.Width });
00155             Point box2Right = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] +
      this.Direction[0] * this.Box2 - perpDirection[0] * this.Width, this.Position[1] + this.Direction[1] *
      this.Box2 - perpDirection[1] * this.Width });
00156             Point whisker2Right = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] +
      this.Direction[0] * this.Whisker2 - perpDirection[0] * this.Width * this.WhiskerWidth,
      this.Position[1] + this.Direction[1] * this.Whisker2 - perpDirection[1] * this.Width *
      this.WhiskerWidth });
00157 
00158             Point medianLeft = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] +
      perpDirection[0] * this.Width, this.Position[1] + perpDirection[1] * this.Width });
00159             Point medianRight = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] -
      perpDirection[0] * this.Width, this.Position[1] - perpDirection[1] * this.Width });
00160 
00161             string whiskerTag = Tag;
00162             string boxFillTag = Tag;
00163             string boxStrokeTag = Tag;
00164             string medianTag = Tag;
00165             string medianSymbolTag = Tag;
00166 
00167             if (!string.IsNullOrEmpty(Tag) && target.UseUniqueTags)
00168             {
00169                 whiskerTag += "@whiskers";
00170                 boxStrokeTag += "@stroke";
00171                 medianTag += "@median";
00172                 medianSymbolTag += "@medianSymbol";
00173             }
00174 
00175             if (NotchSize != 0 && NotchWidth != 1)
00176             {
00177                 medianLeft = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] +
      perpDirection[0] * this.Width * this.NotchWidth, this.Position[1] + perpDirection[1] * this.Width *
      this.NotchWidth });
00178                 medianRight = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0] -
      perpDirection[0] * this.Width * this.NotchWidth, this.Position[1] - perpDirection[1] * this.Width *
      this.NotchWidth });
00179 
00180                 Point notch2Left = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0]
      + this.Direction[0] * this.NotchSize + perpDirection[0] * this.Width, this.Position[1] +
      this.Direction[1] * this.NotchSize + perpDirection[1] * this.Width });
00181                 Point notch1Left = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0]
```

```
- this.Direction[0] * this.NotchSize + perpDirection[0] * this.Width, this.Position[1] -
this.Direction[1] * this.NotchSize + perpDirection[1] * this.Width });
00182
00183                 Point notch2Right = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0]
+ this.Direction[0] * this.NotchSize - perpDirection[0] * this.Width, this.Position[1] +
this.Direction[1] * this.NotchSize - perpDirection[1] * this.Width });
00184                 Point notch1Right = CoordinateSystem.ToPlotCoordinates(new double[] { this.Position[0]
- this.Direction[0] * this.NotchSize - perpDirection[0] * this.Width, this.Position[1] -
this.Direction[1] * this.NotchSize - perpDirection[1] * this.Width });
00185
00186
00187                 GraphicsPath whiskers = new GraphicsPath();
00188
00189                 whiskers.MoveTo(whisker1Left).LineTo(whisker1Right);
00190                 whiskers.MoveTo(whisker1).LineTo(whisker2);
00191                 whiskers.MoveTo(whisker2Left).LineTo(whisker2Right);
00192
00193                 GraphicsPath box = new
GraphicsPath().MoveTo(box1Left).LineTo(box1Right).LineTo(notch1Right).LineTo(medianRight).LineTo(notch2Right).LineTo(box
00194                 GraphicsPath median = new GraphicsPath().MoveTo(medianLeft).LineTo(medianRight);
00195
00196                 if (WhiskersPresentationAttributes.Stroke != null)
00197                 {
00198                     target.StrokePath(whiskers, WhiskersPresentationAttributes.Stroke,
WhiskersPresentationAttributes.LineWidth, WhiskersPresentationAttributes.LineCap,
WhiskersPresentationAttributes.LineJoin, WhiskersPresentationAttributes.LineDash, whiskerTag);
00199                 }
00200
00201                 if (BoxPresentationAttributes.Fill != null)
00202                 {
00203                     target.FillPath(box, BoxPresentationAttributes.Fill, boxFillTag);
00204                 }
00205
00206                 if (BoxPresentationAttributes.Stroke != null)
00207                 {
00208                     target.StrokePath(box, BoxPresentationAttributes.Stroke,
BoxPresentationAttributes.LineWidth, BoxPresentationAttributes.LineCap,
BoxPresentationAttributes.LineJoin, BoxPresentationAttributes.LineDash, boxStrokeTag);
00209                     target.StrokePath(median, BoxPresentationAttributes.Stroke,
BoxPresentationAttributes.LineWidth * 2, BoxPresentationAttributes.LineCap,
BoxPresentationAttributes.LineJoin, BoxPresentationAttributes.LineDash, medianTag);
00210                 }
00211             }
00212             else
00213             {
00214                 GraphicsPath whiskers = new GraphicsPath();
00215
00216                 whiskers.MoveTo(whisker1Left).LineTo(whisker1Right);
00217                 whiskers.MoveTo(whisker1).LineTo(whisker2);
00218                 whiskers.MoveTo(whisker2Left).LineTo(whisker2Right);
00219
00220                 GraphicsPath box = new
GraphicsPath().MoveTo(box1Left).LineTo(box1Right).LineTo(box2Right).LineTo(box2Left).Close();
00221                 GraphicsPath median = new GraphicsPath().MoveTo(medianLeft).LineTo(medianRight);
00222
00223                 if (WhiskersPresentationAttributes.Stroke != null)
00224                 {
00225                     target.StrokePath(whiskers, WhiskersPresentationAttributes.Stroke,
WhiskersPresentationAttributes.LineWidth, WhiskersPresentationAttributes.LineCap,
WhiskersPresentationAttributes.LineJoin, WhiskersPresentationAttributes.LineDash, whiskerTag);
00226                 }
00227
00228                 if (BoxPresentationAttributes.Fill != null)
00229                 {
00230                     target.FillPath(box, BoxPresentationAttributes.Fill, boxFillTag);
00231                 }
00232
00233                 if (BoxPresentationAttributes.Stroke != null)
00234                 {
00235                     target.StrokePath(box, BoxPresentationAttributes.Stroke,
BoxPresentationAttributes.LineWidth, BoxPresentationAttributes.LineCap,
BoxPresentationAttributes.LineJoin, BoxPresentationAttributes.LineDash, boxStrokeTag);
00236                     target.StrokePath(median, BoxPresentationAttributes.Stroke,
BoxPresentationAttributes.LineWidth * 2, LineCaps.Butt, BoxPresentationAttributes.LineJoin,
BoxPresentationAttributes.LineDash, medianTag);
00237                 }
00238             }
00239
00240             if (CentreSymbolPresentationAttributes.Fill != null ||
CentreSymbolPresentationAttributes.Stroke != null)
00241             {
00242                 double medianCircleRadius = Math.Min(Math.Sqrt((medianLeft.X - medianRight.X) *
(medianLeft.X - medianRight.X) + (medianLeft.Y - medianRight.Y) * (medianLeft.Y - medianRight.Y)),
00243                 Math.Min(Math.Sqrt((box1Left.X - box2Left.X) * (box1Left.X - box2Left.X) + (box1Left.Y
- box2Left.Y) * (box1Left.Y - box2Left.Y)),
00244                 Math.Sqrt((box1Right.X - box2Right.X) * (box1Right.X - box2Right.X) + (box1Right.Y -
box2Right.Y) * (box1Right.Y - box2Right.Y)))) * 0.45;
```

```
00245
00246                 target.Save();
00247                 target.Translate((medianLeft.X + medianRight.X) * 0.5, (medianLeft.Y + medianRight.Y)
      * 0.5);
00248                 target.Scale(medianCircleRadius, medianCircleRadius);
00249
00250                 CentreSymbol.Plot(target, CentreSymbolPresentationAttributes, medianSymbolTag);
00251
00252                 target.Restore();
00253             }
00254         }
00255     }
00256 }
```

8.39 CoordinateSystems.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021
00022 namespace VectSharp.Plots
00023 {
00024 /// <summary>
00025 /// Represents a coordinate system.
00026 /// </summary>
00027     public interface ICoordinateSystem
00028     {
00029
00030     }
00031
00032 /// <summary>
00033 /// Represents a coordinate system transforming data points of type <typeparamref name="T"/> into plot
      <see cref="Point"/>s.
00034 /// </summary>
00035 /// <typeparam name="T">The type of data elements.</typeparam>
00036     public interface ICoordinateSystem<T> :  ICoordinateSystem
00037     {
00038 /// <summary>
00039 /// Transform the specified <paramref name="dataPoint"/> into a plot <see cref="Point"/>.
00040 /// </summary>
00041 /// <param name="dataPoint">The data point whose coordinates should be determined.</param>
00042 /// <returns>A <see cref="Point"/> representing the <paramref name="dataPoint"/> in plot
      space.</returns>
00043         Point ToPlotCoordinates(T dataPoint);
00044     }
00045
00046 /// <summary>
00047 /// Represents a coordinate system tranforming data points of type <typeparamref name="T"/> into <see
      langword="double"/>s.
00048 /// </summary>
00049 /// <typeparam name="T">The type of data elements.</typeparam>
00050     public interface ICoordinateSystem1D<T>
00051     {
00052 /// <summary>
00053 /// Transform the specified <paramref name="dataPoint"/> into a <see langword="double"/>.
00054 /// </summary>
00055 /// <param name="dataPoint">The data point whose coordinates should be determined.</param>
00056 /// <returns>A <see langword="double"/> representing the <paramref name="dataPoint"/> in plot
      space.</returns>
00057         double ToPlotCoordinates(T dataPoint);
00058     }
00059
00060 /// <summary>
00061 /// A coordinate system using a custom method to transform data points.
00062 /// </summary>
00063 /// <typeparam name="T">The type of data elements.</typeparam>
```

```
00064     public class CoordinateSystem<T> :  ICoordinateSystem<T>
00065     {
00066 /// <summary>
00067 /// The method used to transform the data elements into plot <see cref="Point"/>s.
00068 /// </summary>
00069         public Func<T, Point> CoordinateFunction { get; set; }
00070 
00071 /// <summary>
00072 /// Create a new <see cref="CoordinateSystem{T}"/> using the specified method.
00073 /// </summary>
00074 /// <param name="coordinateFunction">The method used to transform the data elements into plot <see
      cref="Point"/>s.</param>
00075         public CoordinateSystem(Func<T, Point> coordinateFunction)
00076         {
00077             this.CoordinateFunction = coordinateFunction;
00078         }
00079 
00080 /// <inheritdoc/>
00081         public Point ToPlotCoordinates(T dataPoint)
00082         {
00083             return CoordinateFunction(dataPoint);
00084         }
00085     }
00086 
00087 /// <summary>
00088 /// Represents a coordinate system performing continuous transformations.
00089 /// </summary>
00090     public interface IContinuousCoordinateSystem :  ICoordinateSystem<IReadOnlyList<double»
00091     {
00092 /// <summary>
00093 /// Gets whether the current coordinate system is linear along all directions.
00094 /// </summary>
00095         bool IsLinear { get; }
00096 
00097 /// <summary>
00098 /// Determines whether points aligned along <paramref name="direction"/> in data space are also
      aligned along some line in plot space.
00099 /// </summary>
00100 /// <param name="direction">The direction in data space along which the points should be
      aligned.</param>
00101 /// <returns><see langword="true"/> if any two points aligned along <paramref name="direction"/> in
      data space are also aligned along some line in plot space, <see langword="false"/>
      otherwise.</returns>
00102         bool IsDirectionStraight(IReadOnlyList<double> direction);
00103 
00104 /// <summary>
00105 /// The maximum difference between two points in data space that appear arbitrarily close in plot
      space, or some approximation.
00106 /// </summary>
00107         double[] Resolution { get; }
00108 
00109 /// <summary>
00110 /// Gets a data element that is arbitrarily close to the specified <paramref name="point"/>, along the
      specified <paramref name="direction"/>.
00111 /// </summary>
00112 /// <param name="point">The point close to which the returned data element should be.</param>
00113 /// <param name="direction">The direction (in data space) along which the returned point should
      be.</param>
00114 /// <returns>A data element that is arbitrarily close to the specified <paramref name="point"/>, along
      the specified <paramref name="direction"/>.</returns>
00115         double[] GetAround(IReadOnlyList<double> point, IReadOnlyList<double> direction);
00116     }
00117 
00118 /// <summary>
00119 /// Represents a coordinate system performing continuous invertible transformations.
00120 /// </summary>
00121     public interface IContinuousInvertibleCoordinateSystem :  IContinuousCoordinateSystem
00122     {
00123 /// <summary>
00124 /// Transform a point in plot space back into data space.
00125 /// </summary>
00126 /// <param name="plotPoint">The point in plot space.</param>
00127 /// <returns>The point in data space corresponding to the specified point in plot space.</returns>
00128         double[] ToDataCoordinates(Point plotPoint);
00129     }
00130 
00131 /// <summary>
00132 /// Represents a linear coordinate system.
00133 /// </summary>
00134     public class LinearCoordinateSystem2D :  IContinuousInvertibleCoordinateSystem
00135     {
00136 /// <summary>
00137 /// The minimum X value.
00138 /// </summary>
00139         public double MinX { get; set; }
00140 
00141 /// <summary>
```

```
00142 /// The maximum X value.
00143 /// </summary>
00144         public double MaxX { get; set; }
00145
00146 /// <summary>
00147 /// The minimum Y value.
00148 /// </summary>
00149         public double MinY { get; set; }
00150
00151 /// <summary>
00152 /// The maximum Y value.
00153 /// </summary>
00154         public double MaxY { get; set; }
00155
00156 /// <summary>
00157 /// The X scale.
00158 /// </summary>
00159         public double ScaleX { get; set; }
00160
00161 /// <summary>
00162 /// The y scale.
00163 /// </summary>
00164         public double ScaleY { get; set; }
00165
00166 /// <inheritdoc/>
00167         public bool IsLinear => true;
00168
00169 /// <inheritdoc/>
00170         public bool IsDirectionStraight(IReadOnlyList<double> direction) => true;
00171
00172         private double[] resolution = null;
00173
00174 /// <inheritdoc/>
00175         public double[] Resolution
00176         {
00177             get
00178             {
00179                 return new double[] { (MaxX - MinX) * 0.01, (MaxY - MinY) * 0.01 };
00180             }
00181
00182             set
00183             {
00184                 resolution = value;
00185             }
00186         }
00187
00188 /// <summary>
00189 /// Creates a new <see cref="LinearCoordinateSystem2D"/> manually specifying the parameter values.
00190 /// </summary>
00191 /// <param name="minX">The minimum X value.</param>
00192 /// <param name="maxX">The maximum X value.</param>
00193 /// <param name="minY">The minimum Y value.</param>
00194 /// <param name="maxY">The maximum Y value.</param>
00195 /// <param name="scaleX">The X scale.</param>
00196 /// <param name="scaleY">The Y scale.</param>
00197         public LinearCoordinateSystem2D(double minX, double maxX, double minY, double maxY, double
      scaleX, double scaleY)
00198         {
00199             MinX = minX;
00200             MaxX = maxX;
00201             MinY = minY;
00202             MaxY = maxY;
00203             ScaleX = scaleX;
00204             ScaleY = scaleY;
00205         }
00206
00207 /// <summary>
00208 /// Creates a new <see cref="LinearCoordinateSystem2D"/> determining the value range from the
      <paramref name="data"/>.
00209 /// </summary>
00210 /// <param name="data">The data from which the value range should be determined.</param>
00211 /// <param name="scaleX">The X scale.</param>
00212 /// <param name="scaleY">The Y scale.</param>
00213         public LinearCoordinateSystem2D(IReadOnlyList<IReadOnlyList<double>> data, double scaleX = 350,
      double scaleY = 250)
00214         {
00215             MinX = double.MaxValue;
00216             MinY = double.MaxValue;
00217
00218             MaxX = double.MinValue;
00219             MaxY = double.MinValue;
00220
00221             for (int i = 0; i < data.Count; i++)
00222             {
00223                 MinX = Math.Min(MinX, data[i][0]);
00224                 MinY = Math.Min(MinY, data[i][1]);
00225
```

```
00226                 MaxX = Math.Max(MaxX, data[i][0]);
00227                 MaxY = Math.Max(MaxY, data[i][1]);
00228             }
00229 
00230             double rangeX = MaxX - MinX;
00231             double rangeY = MaxY - MinY;
00232 
00233             MinX -= rangeX * 0.1;
00234             MaxX += rangeX * 0.1;
00235 
00236             MinY -= rangeY * 0.1;
00237             MaxY += rangeY * 0.1;
00238 
00239             ScaleX = scaleX;
00240             ScaleY = scaleY;
00241         }
00242 
00243 /// <summary>
00244 /// Creates a new <see cref="LinearCoordinateSystem2D"/> determining the value range from the
      <paramref name="data"/>.
00245 /// </summary>
00246 /// <param name="data">The data from which the value range should be determined.</param>
00247 /// <param name="scaleX">The X scale.</param>
00248 /// <param name="scaleY">The Y scale.</param>
00249         public LinearCoordinateSystem2D(double[,] data, double scaleX = 350, double scaleY = 250)
00250         {
00251             MinX = double.MaxValue;
00252             MinY = double.MaxValue;
00253 
00254             MaxX = double.MinValue;
00255             MaxY = double.MinValue;
00256 
00257             for (int i = 0; i < data.GetLength(0); i++)
00258             {
00259                 MinX = Math.Min(MinX, data[i, 0]);
00260                 MinY = Math.Min(MinY, data[i, 1]);
00261 
00262                 MaxX = Math.Max(MaxX, data[i, 0]);
00263                 MaxY = Math.Max(MaxY, data[i, 1]);
00264             }
00265 
00266             double rangeX = MaxX - MinX;
00267             double rangeY = MaxY - MinY;
00268 
00269             MinX -= rangeX * 0.1;
00270             MaxX += rangeX * 0.1;
00271 
00272             MinY -= rangeY * 0.1;
00273             MaxY += rangeY * 0.1;
00274 
00275             ScaleX = scaleX;
00276             ScaleY = scaleY;
00277         }
00278 
00279 /// <inheritdoc/>
00280         public double[] ToDataCoordinates(Point plotPoint)
00281         {
00282             return new double[] { plotPoint.X / ScaleX * (MaxX - MinX) + MinX, (ScaleY - plotPoint.Y)
      / ScaleY * (MaxY - MinY) + MinY };
00283         }
00284 
00285 /// <inheritdoc/>
00286         public Point ToPlotCoordinates(IReadOnlyList<double> dataPoint)
00287         {
00288             return new Point((dataPoint[0] - MinX) / (MaxX - MinX) * ScaleX, ScaleY - (dataPoint[1] -
      MinY) / (MaxY - MinY) * ScaleY);
00289         }
00290 
00291 /// <summary>
00292 /// Transforms the specified <paramref name="dataPoint"/> into a plot point.
00293 /// </summary>
00294 /// <param name="dataPoint">The data whose plot coordinates should be determined.</param>
00295 /// <returns>A <see cref="Point"/> representing the <paramref name="dataPoint"/> in plot
      space.</returns>
00296         public Point ToPlotCoordinates(Point dataPoint)
00297         {
00298             return new Point((dataPoint.X - MinX) / (MaxX - MinX) * ScaleX, ScaleY - (dataPoint.Y -
      MinY) / (MaxY - MinY) * ScaleY);
00299         }
00300 
00301 /// <inheritdoc/>
00302         public double[] GetAround(IReadOnlyList<double> point, IReadOnlyList<double> direction)
00303         {
00304             return new double[] { point[0] + direction[0] * this.Resolution.Min(), point[1] +
      direction[1] * this.Resolution.Min() };
00305         }
00306     }
```

```
00307
00308 /// <summary>
00309 /// Represents a logarithmic coordinate system.
00310 /// </summary>
00311     public class LogarithmicCoordinateSystem2D :  IContinuousInvertibleCoordinateSystem
00312     {
00313 /// <summary>
00314 /// The minimum X value (in logarithmic scale).
00315 /// </summary>
00316         public double MinX { get; set; }
00317
00318 /// <summary>
00319 /// The maximum X value (in logarithmic scale).
00320 /// </summary>
00321         public double MaxX { get; set; }
00322
00323 /// <summary>
00324 /// The minimum Y value (in logarithmic scale).
00325 /// </summary>
00326         public double MinY { get; set; }
00327
00328 /// <summary>
00329 /// The maximum Y value (in logarithmic scale).
00330 /// </summary>
00331         public double MaxY { get; set; }
00332
00333 /// <summary>
00334 /// The X scale.
00335 /// </summary>
00336         public double ScaleX { get; set; }
00337
00338 /// <summary>
00339 /// The y scale.
00340 /// </summary>
00341         public double ScaleY { get; set; }
00342
00343 /// <inheritdoc/>
00344         public bool IsLinear => false;
00345
00346 /// <inheritdoc/>
00347         public bool IsDirectionStraight(IReadOnlyList<double> direction)
00348         {
00349             if (direction[0] == 0 || direction[1] == 0)
00350             {
00351                 return true;
00352             }
00353             else
00354             {
00355                 return false;
00356             }
00357         }
00358
00359         private double[] resolution = null;
00360
00361 /// <inheritdoc/>
00362         public double[] Resolution
00363         {
00364             get
00365             {
00366                 if (resolution == null)
00367                 {
00368                     return new double[] { (Math.Exp(MaxX) - Math.Exp(MinX)) * 0.01, (Math.Exp(MaxY) -
      Math.Exp(MinY)) * 0.01 };
00369                 }
00370                 else
00371                 {
00372                     return resolution;
00373                 }
00374             }
00375
00376             set
00377             {
00378                 resolution = value;
00379             }
00380         }
00381
00382 /// <summary>
00383 /// Creates a new <see cref="LogarithmicCoordinateSystem2D"/> manually specifying the parameter
      values.
00384 /// </summary>
00385 /// <param name="minX">The minimum X value.</param>
00386 /// <param name="maxX">The maximum X value.</param>
00387 /// <param name="minY">The minimum Y value.</param>
00388 /// <param name="maxY">The maximum Y value.</param>
00389 /// <param name="scaleX">The X scale.</param>
00390 /// <param name="scaleY">The Y scale.</param>
00391         public LogarithmicCoordinateSystem2D(double minX, double maxX, double minY, double maxY,
```

```
      double scaleX, double scaleY)
00392         {
00393             MinX = Math.Log(minX);
00394             MaxX = Math.Log(maxX);
00395             MinY = Math.Log(minY);
00396             MaxY = Math.Log(maxY);
00397             ScaleX = scaleX;
00398             ScaleY = scaleY;
00399         }
00400
00401 /// <summary>
00402 /// Creates a new <see cref="LogarithmicCoordinateSystem2D"/> determining the value range from the
      <paramref name="data"/>.
00403 /// </summary>
00404 /// <param name="data">The data from which the value range should be determined.</param>
00405 /// <param name="scaleX">The X scale.</param>
00406 /// <param name="scaleY">The Y scale.</param>
00407         public LogarithmicCoordinateSystem2D(IReadOnlyList<IReadOnlyList<double» data, double scaleX =
      350, double scaleY = 250)
00408         {
00409             MinX = double.MaxValue;
00410             MinY = double.MaxValue;
00411
00412             MaxX = double.MinValue;
00413             MaxY = double.MinValue;
00414
00415             for (int i = 0; i < data.Count; i++)
00416             {
00417                 MinX = Math.Min(MinX, data[i][0]);
00418                 MinY = Math.Min(MinY, data[i][1]);
00419
00420                 MaxX = Math.Max(MaxX, data[i][0]);
00421                 MaxY = Math.Max(MaxY, data[i][1]);
00422             }
00423
00424             MinX = Math.Log(MinX);
00425             MinY = Math.Log(MinY);
00426             MaxX = Math.Log(MaxX);
00427             MaxY = Math.Log(MaxY);
00428
00429             double rangeX = MaxX - MinX;
00430             double rangeY = MaxY - MinY;
00431
00432             MinX -= rangeX * 0.1;
00433             MaxX += rangeX * 0.1;
00434
00435             MinY -= rangeY * 0.1;
00436             MaxY += rangeY * 0.1;
00437
00438             ScaleX = scaleX;
00439             ScaleY = scaleY;
00440         }
00441
00442
00443 /// <summary>
00444 /// Creates a new <see cref="LogarithmicCoordinateSystem2D"/> determining the value range from the
      <paramref name="data"/>.
00445 /// </summary>
00446 /// <param name="data">The data from which the value range should be determined.</param>
00447 /// <param name="scaleX">The X scale.</param>
00448 /// <param name="scaleY">The Y scale.</param>
00449         public LogarithmicCoordinateSystem2D(double[,] data, double scaleX = 350, double scaleY = 250)
00450         {
00451             MinX = double.MaxValue;
00452             MinY = double.MaxValue;
00453
00454             MaxX = double.MinValue;
00455             MaxY = double.MinValue;
00456
00457             for (int i = 0; i < data.GetLength(0); i++)
00458             {
00459                 MinX = Math.Min(MinX, data[i, 0]);
00460                 MinY = Math.Min(MinY, data[i, 1]);
00461
00462                 MaxX = Math.Max(MaxX, data[i, 0]);
00463                 MaxY = Math.Max(MaxY, data[i, 1]);
00464             }
00465
00466             MinX = Math.Log(MinX);
00467             MinY = Math.Log(MinY);
00468             MaxX = Math.Log(MaxX);
00469             MaxY = Math.Log(MaxY);
00470
00471             double rangeX = MaxX - MinX;
00472             double rangeY = MaxY - MinY;
00473
00474             MinX -= rangeX * 0.1;
```

```
00475             MaxX += rangeX * 0.1;
00476
00477             MinY -= rangeY * 0.1;
00478             MaxY += rangeY * 0.1;
00479
00480             ScaleX = scaleX;
00481             ScaleY = scaleY;
00482         }
00483
00484 /// <inheritdoc/>
00485         public double[] ToDataCoordinates(Point plotPoint)
00486         {
00487             return new double[] { Math.Exp(plotPoint.X / ScaleX * (MaxX - MinX) + MinX),
      Math.Exp((ScaleY - plotPoint.Y) / ScaleY * (MaxY - MinY) + MinY) };
00488         }
00489
00490 /// <inheritdoc/>
00491         public Point ToPlotCoordinates(IReadOnlyList<double> dataPoint)
00492         {
00493             return new Point((Math.Log(dataPoint[0]) - MinX) / (MaxX - MinX) * ScaleX, ScaleY -
      (Math.Log(dataPoint[1]) - MinY) / (MaxY - MinY) * ScaleY);
00494         }
00495
00496 /// <summary>
00497 /// Transforms the specified <paramref name="dataPoint"/> into a plot point.
00498 /// </summary>
00499 /// <param name="dataPoint">The data whose plot coordinates should be determined.</param>
00500 /// <returns>A <see cref="Point"/> representing the <paramref name="dataPoint"/> in plot
      space.</returns>
00501         public Point ToPlotCoordinates(Point dataPoint)
00502         {
00503             return new Point((Math.Log(dataPoint.X) - MinX) / (MaxX - MinX) * ScaleX, ScaleY -
      (Math.Log(dataPoint.Y) - MinY) / (MaxY - MinY) * ScaleY);
00504         }
00505
00506 /// <inheritdoc/>
00507         public double[] GetAround(IReadOnlyList<double> point, IReadOnlyList<double> direction)
00508         {
00509             return new double[] { Math.Exp(Math.Log(point[0]) + direction[0] * (MaxX - MinX) * 0.01),
      Math.Exp(Math.Log(point[1]) + direction[1] * (MaxY - MinY) * 0.01) };
00510         }
00511     }
00512
00513 /// <summary>
00514 /// Represents a semi-logarithmic coordinate system with a logarithmic transformation on the Y axis.
00515 /// </summary>
00516     public class LogLinCoordinateSystem2D :  IContinuousInvertibleCoordinateSystem
00517     {
00518 /// <summary>
00519 /// The minimum X value.
00520 /// </summary>
00521         public double MinX { get; set; }
00522
00523 /// <summary>
00524 /// The maximum X value.
00525 /// </summary>
00526         public double MaxX { get; set; }
00527
00528 /// <summary>
00529 /// The minimum Y value (in logarithmic scale).
00530 /// </summary>
00531         public double MinY { get; set; }
00532
00533 /// <summary>
00534 /// The maximum Y value (in logarithmic scale).
00535 /// </summary>
00536         public double MaxY { get; set; }
00537
00538 /// <summary>
00539 /// The X scale.
00540 /// </summary>
00541         public double ScaleX { get; set; }
00542
00543 /// <summary>
00544 /// The y scale.
00545 /// </summary>
00546         public double ScaleY { get; set; }
00547
00548 /// <inheritdoc/>
00549         public bool IsLinear => false;
00550
00551 /// <inheritdoc/>
00552         public bool IsDirectionStraight(IReadOnlyList<double> direction)
00553         {
00554             if (direction[0] == 0 || direction[1] == 0)
00555             {
00556                 return true;
```

```
00557             }
00558             else
00559             {
00560                 return false;
00561             }
00562         }
00563 
00564         private double[] resolution = null;
00565 
00566 /// <inheritdoc/>
00567         public double[] Resolution
00568         {
00569             get
00570             {
00571                 if (resolution == null)
00572                 {
00573                     return new double[] { (MaxX - MinX) * 0.01, (Math.Exp(MaxY) - Math.Exp(MinY)) *
      0.01 };
00574                 }
00575                 else
00576                 {
00577                     return resolution;
00578                 }
00579             }
00580 
00581             set
00582             {
00583                 resolution = value;
00584             }
00585         }
00586 
00587 /// <summary>
00588 /// Creates a new <see cref="LogLinCoordinateSystem2D"/> manually specifying the parameter values.
00589 /// </summary>
00590 /// <param name="minX">The minimum X value.</param>
00591 /// <param name="maxX">The maximum X value.</param>
00592 /// <param name="minY">The minimum Y value.</param>
00593 /// <param name="maxY">The maximum Y value.</param>
00594 /// <param name="scaleX">The X scale.</param>
00595 /// <param name="scaleY">The Y scale.</param>
00596         public LogLinCoordinateSystem2D(double minX, double maxX, double minY, double maxY, double
      scaleX, double scaleY)
00597         {
00598             MinX = minX;
00599             MaxX = maxX;
00600             MinY = Math.Log(minY);
00601             MaxY = Math.Log(maxY);
00602             ScaleX = scaleX;
00603             ScaleY = scaleY;
00604         }
00605 
00606 /// <summary>
00607 /// Creates a new <see cref="LogLinCoordinateSystem2D"/> determining the value range from the
      <paramref name="data"/>.
00608 /// </summary>
00609 /// <param name="data">The data from which the value range should be determined.</param>
00610 /// <param name="scaleX">The X scale.</param>
00611 /// <param name="scaleY">The Y scale.</param>
00612         public LogLinCoordinateSystem2D(IReadOnlyList<IReadOnlyList<double» data, double scaleX = 350,
      double scaleY = 250)
00613         {
00614             MinX = double.MaxValue;
00615             MinY = double.MaxValue;
00616 
00617             MaxX = double.MinValue;
00618             MaxY = double.MinValue;
00619 
00620             for (int i = 0; i < data.Count; i++)
00621             {
00622                 MinX = Math.Min(MinX, data[i][0]);
00623                 MinY = Math.Min(MinY, data[i][1]);
00624 
00625                 MaxX = Math.Max(MaxX, data[i][0]);
00626                 MaxY = Math.Max(MaxY, data[i][1]);
00627             }
00628 
00629             MinX = MinX;
00630             MinY = Math.Log(MinY);
00631             MaxX = MaxX;
00632             MaxY = Math.Log(MaxY);
00633 
00634             double rangeX = MaxX - MinX;
00635             double rangeY = MaxY - MinY;
00636 
00637             MinX -= rangeX * 0.1;
00638             MaxX += rangeX * 0.1;
00639 
```

```
00640             MinY -= rangeY * 0.1;
00641             MaxY += rangeY * 0.1;
00642 
00643             ScaleX = scaleX;
00644             ScaleY = scaleY;
00645         }
00646 
00647 /// <summary>
00648 /// Creates a new <see cref="LogLinCoordinateSystem2D"/> determining the value range from the
      <paramref name="data"/>.
00649 /// </summary>
00650 /// <param name="data">The data from which the value range should be determined.</param>
00651 /// <param name="scaleX">The X scale.</param>
00652 /// <param name="scaleY">The Y scale.</param>
00653         public LogLinCoordinateSystem2D(double[,] data, double scaleX = 350, double scaleY = 250)
00654         {
00655             MinX = double.MaxValue;
00656             MinY = double.MaxValue;
00657 
00658             MaxX = double.MinValue;
00659             MaxY = double.MinValue;
00660 
00661             for (int i = 0; i < data.GetLength(0); i++)
00662             {
00663                 MinX = Math.Min(MinX, data[i, 0]);
00664                 MinY = Math.Min(MinY, data[i, 1]);
00665 
00666                 MaxX = Math.Max(MaxX, data[i, 0]);
00667                 MaxY = Math.Max(MaxY, data[i, 1]);
00668             }
00669 
00670             MinX = MinX;
00671             MinY = Math.Log(MinY);
00672             MaxX = MaxX;
00673             MaxY = Math.Log(MaxY);
00674 
00675             double rangeX = MaxX - MinX;
00676             double rangeY = MaxY - MinY;
00677 
00678             MinX -= rangeX * 0.1;
00679             MaxX += rangeX * 0.1;
00680 
00681             MinY -= rangeY * 0.1;
00682             MaxY += rangeY * 0.1;
00683 
00684             ScaleX = scaleX;
00685             ScaleY = scaleY;
00686         }
00687 
00688 /// <inheritdoc/>
00689         public double[] ToDataCoordinates(Point plotPoint)
00690         {
00691             return new double[] { plotPoint.X / ScaleX * (MaxX - MinX) + MinX, Math.Exp((ScaleY -
      plotPoint.Y) / ScaleY * (MaxY - MinY) + MinY) };
00692         }
00693 
00694 /// <inheritdoc/>
00695         public Point ToPlotCoordinates(IReadOnlyList<double> dataPoint)
00696         {
00697             return new Point((dataPoint[0] - MinX) / (MaxX - MinX) * ScaleX, ScaleY -
      (Math.Log(dataPoint[1]) - MinY) / (MaxY - MinY) * ScaleY);
00698         }
00699 
00700 /// <summary>
00701 /// Transforms the specified <paramref name="dataPoint"/> into a plot point.
00702 /// </summary>
00703 /// <param name="dataPoint">The data whose plot coordinates should be determined.</param>
00704 /// <returns>A <see cref="Point"/> representing the <paramref name="dataPoint"/> in plot
      space.</returns>
00705         public Point ToPlotCoordinates(Point dataPoint)
00706         {
00707             return new Point((dataPoint.X - MinX) / (MaxX - MinX) * ScaleX, ScaleY -
      (Math.Log(dataPoint.Y) - MinY) / (MaxY - MinY) * ScaleY);
00708         }
00709 
00710 /// <inheritdoc/>
00711         public double[] GetAround(IReadOnlyList<double> point, IReadOnlyList<double> direction)
00712         {
00713             return new double[] { point[0] + direction[0] * (MaxX - MinX) * 0.01,
      Math.Exp(Math.Log(point[1]) + direction[1] * (MaxY - MinY) * 0.01) };
00714         }
00715     }
00716 
00717 /// <summary>
00718 /// Represents a semi-logarithmic coordinate system with a logarithmic transformation on the X axis.
00719 /// </summary>
00720     public class LinLogCoordinateSystem2D :  IContinuousInvertibleCoordinateSystem
```

```
00721     {
00722 /// <summary>
00723 /// The minimum X value.
00724 /// </summary>
00725         public double MinX { get; set; }
00726 
00727 /// <summary>
00728 /// The maximum X value.
00729 /// </summary>
00730         public double MaxX { get; set; }
00731 
00732 /// <summary>
00733 /// The minimum Y value (in logarithmic scale).
00734 /// </summary>
00735         public double MinY { get; set; }
00736 
00737 /// <summary>
00738 /// The maximum Y value (in logarithmic scale).
00739 /// </summary>
00740         public double MaxY { get; set; }
00741 
00742 /// <summary>
00743 /// The X scale.
00744 /// </summary>
00745         public double ScaleX { get; set; }
00746 
00747 /// <summary>
00748 /// The y scale.
00749 /// </summary>
00750         public double ScaleY { get; set; }
00751 
00752 /// <inheritdoc/>
00753         public bool IsLinear => false;
00754 
00755 /// <inheritdoc/>
00756         public bool IsDirectionStraight(IReadOnlyList<double> direction)
00757         {
00758             if (direction[0] == 0 || direction[1] == 0)
00759             {
00760                 return true;
00761             }
00762             else
00763             {
00764                 return false;
00765             }
00766         }
00767 
00768         private double[] resolution = null;
00769 
00770 /// <inheritdoc/>
00771         public double[] Resolution
00772         {
00773             get
00774             {
00775                 if (resolution == null)
00776                 {
00777                     return new double[] { (MaxX - MinX) * 0.01, (Math.Exp(MaxY) - Math.Exp(MinY)) *
      0.01 };
00778                 }
00779                 else
00780                 {
00781                     return resolution;
00782                 }
00783             }
00784 
00785             set
00786             {
00787                 resolution = value;
00788             }
00789         }
00790 
00791 /// <summary>
00792 /// Creates a new <see cref="LinLogCoordinateSystem2D"/> manually specifying the parameter values.
00793 /// </summary>
00794 /// <param name="minX">The minimum X value.</param>
00795 /// <param name="maxX">The maximum X value.</param>
00796 /// <param name="minY">The minimum Y value.</param>
00797 /// <param name="maxY">The maximum Y value.</param>
00798 /// <param name="scaleX">The X scale.</param>
00799 /// <param name="scaleY">The Y scale.</param>
00800         public LinLogCoordinateSystem2D(double minX, double maxX, double minY, double maxY, double
      scaleX, double scaleY)
00801         {
00802             MinX = Math.Log(minX);
00803             MaxX = Math.Log(maxX);
00804             MinY = minY;
00805             MaxY = maxY;
```

```
00806             ScaleX = scaleX;
00807             ScaleY = scaleY;
00808         }
00809
00810 /// <summary>
00811 /// Creates a new <see cref="LinLogCoordinateSystem2D"/> determining the value range from the
      <paramref name="data"/>.
00812 /// </summary>
00813 /// <param name="data">The data from which the value range should be determined.</param>
00814 /// <param name="scaleX">The X scale.</param>
00815 /// <param name="scaleY">The Y scale.</param>
00816         public LinLogCoordinateSystem2D(IReadOnlyList<IReadOnlyList<double» data, double scaleX = 350,
      double scaleY = 250)
00817         {
00818             MinX = double.MaxValue;
00819             MinY = double.MaxValue;
00820
00821             MaxX = double.MinValue;
00822             MaxY = double.MinValue;
00823
00824             for (int i = 0; i < data.Count; i++)
00825             {
00826                 MinX = Math.Min(MinX, data[i][0]);
00827                 MinY = Math.Min(MinY, data[i][1]);
00828
00829                 MaxX = Math.Max(MaxX, data[i][0]);
00830                 MaxY = Math.Max(MaxY, data[i][1]);
00831             }
00832
00833             MinX = Math.Log(MinX);
00834             MinY = MinY;
00835             MaxX = Math.Log(MaxX);
00836             MaxY = MaxY;
00837
00838             double rangeX = MaxX - MinX;
00839             double rangeY = MaxY - MinY;
00840
00841             MinX -= rangeX * 0.1;
00842             MaxX += rangeX * 0.1;
00843
00844             MinY -= rangeY * 0.1;
00845             MaxY += rangeY * 0.1;
00846
00847             ScaleX = scaleX;
00848             ScaleY = scaleY;
00849         }
00850
00851 /// <summary>
00852 /// Creates a new <see cref="LinLogCoordinateSystem2D"/> determining the value range from the
      <paramref name="data"/>.
00853 /// </summary>
00854 /// <param name="data">The data from which the value range should be determined.</param>
00855 /// <param name="scaleX">The X scale.</param>
00856 /// <param name="scaleY">The Y scale.</param>
00857         public LinLogCoordinateSystem2D(double[,] data, double scaleX = 350, double scaleY = 250)
00858         {
00859             MinX = double.MaxValue;
00860             MinY = double.MaxValue;
00861
00862             MaxX = double.MinValue;
00863             MaxY = double.MinValue;
00864
00865             for (int i = 0; i < data.GetLength(0); i++)
00866             {
00867                 MinX = Math.Min(MinX, data[i, 0]);
00868                 MinY = Math.Min(MinY, data[i, 1]);
00869
00870                 MaxX = Math.Max(MaxX, data[i, 0]);
00871                 MaxY = Math.Max(MaxY, data[i, 1]);
00872             }
00873
00874             MinX = Math.Log(MinX);
00875             MinY = MinY;
00876             MaxX = Math.Log(MaxX);
00877             MaxY = MaxY;
00878
00879             double rangeX = MaxX - MinX;
00880             double rangeY = MaxY - MinY;
00881
00882             MinX -= rangeX * 0.1;
00883             MaxX += rangeX * 0.1;
00884
00885             MinY -= rangeY * 0.1;
00886             MaxY += rangeY * 0.1;
00887
00888             ScaleX = scaleX;
00889             ScaleY = scaleY;
```

```
00890         }
00891
00892 /// <inheritdoc/>
00893         public double[] ToDataCoordinates(Point plotPoint)
00894         {
00895             return new double[] { Math.Exp(plotPoint.X / ScaleX * (MaxX - MinX) + MinX), (ScaleY -
      plotPoint.Y) / ScaleY * (MaxY - MinY) + MinY };
00896         }
00897
00898 /// <inheritdoc/>
00899         public Point ToPlotCoordinates(IReadOnlyList<double> dataPoint)
00900         {
00901             return new Point((Math.Log(dataPoint[0]) - MinX) / (MaxX - MinX) * ScaleX, ScaleY -
      (dataPoint[1] - MinY) / (MaxY - MinY) * ScaleY);
00902         }
00903
00904 /// <summary>
00905 /// Transforms the specified <paramref name="dataPoint"/> into a plot point.
00906 /// </summary>
00907 /// <param name="dataPoint">The data whose plot coordinates should be determined.</param>
00908 /// <returns>A <see cref="Point"/> representing the <paramref name="dataPoint"/> in plot
      space.</returns>
00909         public Point ToPlotCoordinates(Point dataPoint)
00910         {
00911             return new Point((Math.Log(dataPoint.X) - MinX) / (MaxX - MinX) * ScaleX, ScaleY -
      (dataPoint.Y - MinY) / (MaxY - MinY) * ScaleY);
00912         }
00913
00914 /// <inheritdoc/>
00915         public double[] GetAround(IReadOnlyList<double> point, IReadOnlyList<double> direction)
00916         {
00917             return new double[] { Math.Exp(Math.Log(point[0]) + direction[0] * (MaxX - MinX) * 0.01),
      point[1] + direction[1] * (MaxY - MinY) * 0.01 };
00918         }
00919     }
00920
00921 /// <summary>
00922 /// Represents a 1-D linear coordinate system.
00923 /// </summary>
00924     public class LinearCoordinateSystem1D :  ICoordinateSystem1D<double>
00925     {
00926 /// <summary>
00927 /// The minimum value.
00928 /// </summary>
00929         public double Min { get; set; }
00930
00931 /// <summary>
00932 /// The maximum value.
00933 /// </summary>
00934         public double Max { get; set; }
00935
00936 /// <summary>
00937 /// The scale factor.
00938 /// </summary>
00939         public double Scale { get; set; }
00940
00941 /// <summary>
00942 /// Creates a new <see cref="LinearCoordinateSystem1D"/>, manually specifying the parameters.
00943 /// </summary>
00944 /// <param name="min">The minimum value.</param>
00945 /// <param name="max">The maximum value.</param>
00946 /// <param name="scale">The scale factor.</param>
00947         public LinearCoordinateSystem1D(double min, double max, double scale)
00948         {
00949             Min = min;
00950             Max = max;
00951             Scale = scale;
00952         }
00953
00954 /// <summary>
00955 /// Creates a new <see cref="LinearCoordinateSystem1D"/> determining the value range from the
      <paramref name="data"/>.
00956 /// </summary>
00957 /// <param name="data">The data from which the value range should be determined.</param>
00958 /// <param name="scale">The scale factor.</param>
00959         public LinearCoordinateSystem1D(IReadOnlyList<double> data, double scale = 350)
00960         {
00961             Min = double.MaxValue;
00962             Min = double.MaxValue;
00963
00964             for (int i = 0; i < data.Count; i++)
00965             {
00966                 Min = Math.Min(Min, data[i]);
00967                 Max = Math.Max(Max, data[i]);
00968             }
00969
00970             double range = Max - Min;
```

```
00971
00972             Min -= range * 0.1;
00973             Max += range * 0.1;
00974
00975             Scale = scale;
00976         }
00977
00978 /// <inheritdoc/>
00979         public double ToPlotCoordinates(double dataPoint)
00980         {
00981             return (dataPoint - Min) / (Max - Min) * Scale;
00982         }
00983     }
00984
00985 /// <summary>
00986 /// Represents a 1-D logarithmic coordinate system.
00987 /// </summary>
00988     public class LogarithmicCoordinateSystem1D :  ICoordinateSystem1D<double>
00989     {
00990 /// <summary>
00991 /// The minimum value (in logarithmic scale).
00992 /// </summary>
00993         public double Min { get; set; }
00994
00995 /// <summary>
00996 /// The maximum value (in logarithmic scale).
00997 /// </summary>
00998         public double Max { get; set; }
00999
01000 /// <summary>
01001 /// The scale factor.
01002 /// </summary>
01003         public double Scale { get; set; }
01004
01005 /// <summary>
01006 /// Creates a new <see cref="LogarithmicCoordinateSystem1D"/>, manually specifying the parameters.
01007 /// </summary>
01008 /// <param name="min">The minimum value.</param>
01009 /// <param name="max">The maximum value.</param>
01010 /// <param name="scale">The scale factor.</param>
01011         public LogarithmicCoordinateSystem1D(double min, double max, double scale)
01012         {
01013             Min = Math.Log(min);
01014             Max = Math.Log(max);
01015             Scale = scale;
01016         }
01017
01018 /// <summary>
01019 /// Creates a new <see cref="LogarithmicCoordinateSystem1D"/> determining the value range from the
      <paramref name="data"/>.
01020 /// </summary>
01021 /// <param name="data">The data from which the value range should be determined.</param>
01022 /// <param name="scale">The scale factor.</param>
01023         public LogarithmicCoordinateSystem1D(IReadOnlyList<double> data, double scale = 350)
01024         {
01025             Min = double.MaxValue;
01026             Min = double.MaxValue;
01027
01028             for (int i = 0; i < data.Count; i++)
01029             {
01030                 Min = Math.Min(Min, data[i]);
01031                 Max = Math.Max(Max, data[i]);
01032             }
01033
01034             Min = Math.Log(Min);
01035             Max = Math.Log(Max);
01036
01037             double range = Max - Min;
01038
01039             Min -= range * 0.1;
01040             Max += range * 0.1;
01041
01042             Scale = scale;
01043         }
01044
01045 /// <inheritdoc/>
01046         public double ToPlotCoordinates(double dataPoint)
01047         {
01048             return (Math.Log(dataPoint) - Min) / (Max - Min) * Scale;
01049         }
01050     }
01051
01052 /// <summary>
01053 /// A coordinate system using a custom method to transform data points.
01054 /// </summary>
01055 /// <typeparam name="T">The type of data elements.</typeparam>
01056     public class CoordinateSystem1D<T> :  ICoordinateSystem1D<T>
```

```
01057     {
01058 /// <summary>
01059 /// The method used to transform data points.
01060 /// </summary>
01061         public Func<T, double> CoordinateFunction { get; set; }
01062 
01063 /// <summary>
01064 /// Creates a new <see cref="CoordinateSystem1D{T}"/> using the specified <paramref
      name="coordinateFunction"/>.
01065 /// </summary>
01066 /// <param name="coordinateFunction">The method used to transform data points.</param>
01067         public CoordinateSystem1D(Func<T, double> coordinateFunction)
01068         {
01069             this.CoordinateFunction = coordinateFunction;
01070         }
01071 
01072 /// <inheritdoc/>
01073         public double ToPlotCoordinates(T dataPoint)
01074         {
01075             return CoordinateFunction(dataPoint);
01076         }
01077     }
01078 
01079 /// <summary>
01080 /// Represents a categorical 1-D coordinate system.
01081 /// </summary>
01082 /// <typeparam name="T">The type of data elements.</typeparam>
01083     public class CategoricalCoordinateSystem1D<T> :  ICoordinateSystem1D<T>
01084     {
01085 /// <summary>
01086 /// A <see cref="Dictionary{TKey, TValue}"/> storing the coordinates.
01087 /// </summary>
01088         public Dictionary<T, double> Coordinates { get; set; }
01089 
01090 /// <summary>
01091 /// Creates a new <see cref="CategoricalCoordinateSystem1D{T}"/> using the coordinates specified in
      the supplied <paramref name="coordinates"/> <see cref="Dictionary{TKey, TValue}"/>.
01092 /// </summary>
01093 /// <param name="coordinates"></param>
01094         public CategoricalCoordinateSystem1D(Dictionary<T, double> coordinates)
01095         {
01096             this.Coordinates = coordinates;
01097         }
01098 
01099 /// <summary>
01100 /// Creates a new <see cref="CategoricalCoordinateSystem1D{T}"/> assigning equally-spaced coordinates
      to all different values of <paramref name="data"/>.
01101 /// </summary>
01102 /// <param name="data">The data containing discrete parameter values that should be assigned to
      different coordinates.</param>
01103 /// <param name="scale">The maximum coordinate.</param>
01104         public CategoricalCoordinateSystem1D(IReadOnlyList<T> data, double scale = 350)
01105         {
01106             Coordinates = new Dictionary<T, double>();
01107 
01108             for (int i = 0; i < data.Count; i++)
01109             {
01110                 if (!Coordinates.ContainsKey(data[i]))
01111                 {
01112                     Coordinates[data[i]] = Coordinates.Count;
01113                 }
01114             }
01115 
01116             if (Coordinates.Count > 1)
01117             {
01118                 foreach (KeyValuePair<T, double> kvp in Coordinates)
01119                 {
01120                     Coordinates[kvp.Key] = kvp.Value / (Coordinates.Count - 1) * scale; ;
01121                 }
01122             }
01123         }
01124 
01125 /// <inheritdoc/>
01126         public double ToPlotCoordinates(T dataPoint)
01127         {
01128             return Coordinates[dataPoint];
01129         }
01130     }
01131 
01132 /// <summary>
01133 /// Combines two <see cref="ICoordinateSystem1D{T}"/>s to produce a <see
      cref="ICoordinateSystem{T}"/>.
01134 /// </summary>
01135 /// <typeparam name="T1">The type for the first coordinate system.</typeparam>
01136 /// <typeparam name="T2">The type for the second coordinate system.</typeparam>
01137     public class CompositeCoordinateSystem2D<T1, T2> :  ICoordinateSystem<(T1, T2)>
01138     {
```

```
01139 /// <summary>
01140 /// The first coordinate system.
01141 /// </summary>
01142         public ICoordinateSystem1D<T1> CoordinateSystemX { get; set; }
01143 
01144 /// <summary>
01145 /// The second coordinate system.
01146 /// </summary>
01147         public ICoordinateSystem1D<T2> CoordinateSystemY { get; set; }
01148 
01149 /// <summary>
01150 /// Creates a new <see cref="CompositeCoordinateSystem2D{T1, T2}"/> from two 1-D coordinate system.
01151 /// </summary>
01152 /// <param name="coordinateSystemX">The first coordinate system.</param>
01153 /// <param name="coordinateSystemY">The second coordinate system.</param>
01154         public CompositeCoordinateSystem2D(ICoordinateSystem1D<T1> coordinateSystemX,
      ICoordinateSystem1D<T2> coordinateSystemY)
01155         {
01156             this.CoordinateSystemX = coordinateSystemX;
01157             this.CoordinateSystemY = coordinateSystemY;
01158         }
01159 
01160 /// <inheritdoc/>
01161         public Point ToPlotCoordinates((T1, T2) dataPoint)
01162         {
01163             return new Point(CoordinateSystemX.ToPlotCoordinates(dataPoint.Item1),
      CoordinateSystemY.ToPlotCoordinates(dataPoint.Item2));
01164         }
01165     }
01166 }
```

8.40 DataPoints.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
00019 using System.Collections.Generic;
00020 
00021 namespace VectSharp.Plots
00022 {
00023 /// <summary>
00024 /// Represents a symbol that can be added to the plot at a specified position.
00025 /// </summary>
00026     public interface IDataPointElement
00027     {
00028 /// <summary>
00029 /// Draw the symbol on the plot.
00030 /// </summary>
00031 /// <param name="target">The <see cref="Graphics"/> object on which to draw.  It is assumed that it
      has been transformed so that the symbol can be drawn centred at (0, 0)</param>
00032 /// <param name="presentationAttributes">Presentation attributes determining the appearance of the
      symbol.</param>
00033 /// <param name="tag">A tag to identify the symbol in the plot.</param>
00034         void Plot(Graphics target, PlotElementPresentationAttributes presentationAttributes, string
      tag);
00035     }
00036 
00037 /// <summary>
00038 /// A symbol defined by a <see cref="GraphicsPath"/>.
00039 /// </summary>
00040     public class PathDataPointElement :  IDataPointElement
00041     {
00042 /// <summary>
00043 /// The <see cref="GraphicsPath"/> that constitutes the symbol (by default, a circle).
00044 /// </summary>
00045         public GraphicsPath Path { get; set; } = new GraphicsPath().Arc(0, 0, 1, 0, 2 *
      Math.PI).Close();
00046 
```

```
00047 /// <inheritdoc/>
00048         public void Plot(Graphics target, PlotElementPresentationAttributes presentationAttributes,
      string tag)
00049         {
00050             if (string.IsNullOrEmpty(tag))
00051             {
00052                 if (presentationAttributes.Fill != null)
00053                 {
00054                     target.FillPath(Path, presentationAttributes.Fill);
00055                 }
00056 
00057                 if (presentationAttributes.Stroke != null)
00058                 {
00059                     target.StrokePath(Path, presentationAttributes.Stroke,
      presentationAttributes.LineWidth, presentationAttributes.LineCap, presentationAttributes.LineJoin,
      presentationAttributes.LineDash);
00060                 }
00061             }
00062             else
00063             {
00064                 if (presentationAttributes.Fill != null)
00065                 {
00066                     target.FillPath(Path, presentationAttributes.Fill, tag);
00067                 }
00068 
00069                 if (presentationAttributes.Stroke != null)
00070                 {
00071                     target.StrokePath(Path, presentationAttributes.Stroke,
      presentationAttributes.LineWidth, presentationAttributes.LineCap, presentationAttributes.LineJoin,
      presentationAttributes.LineDash, tag + (target.UseUniqueTags ?  "@stroke" :  ""));
00072                 }
00073             }
00074         }
00075 
00076 /// <summary>
00077 /// Create a new <see cref="PathDataPointElement"/> instance representing a circle.
00078 /// </summary>
00079         public PathDataPointElement()
00080         {
00081 
00082         }
00083 
00084 /// <summary>
00085 /// Create a new <see cref="PathDataPointElement"/> instance with the specified <paramref
      name="path"/>.
00086 /// </summary>
00087         public PathDataPointElement(GraphicsPath path)
00088         {
00089             this.Path = path;
00090         }
00091     }
00092 
00093 /// <summary>
00094 /// A symbol defined by a <see cref="VectSharp.Graphics"/> object.
00095 /// </summary>
00096     public class GraphicsDataPointElement :  IDataPointElement
00097     {
00098 /// <summary>
00099 /// The <see cref="VectSharp.Graphics"/> object that will be copied on the plot.
00100 /// </summary>
00101         public Graphics Graphics { get; set; }
00102 
00103 /// <inheritdoc/>
00104         public void Plot(Graphics target, PlotElementPresentationAttributes presentationAttributes,
      string tag)
00105         {
00106             target.DrawGraphics(0, 0, Graphics, tag);
00107         }
00108 
00109 /// <summary>
00110 /// Creates a new <see cref="GraphicsDataPointElement"/> instance.
00111 /// </summary>
00112 /// <param name="graphics"></param>
00113         public GraphicsDataPointElement(Graphics graphics)
00114         {
00115             Graphics = graphics;
00116         }
00117     }
00118 
00119 /// <summary>
00120 /// A symbol drawn by a custom <see cref="Action"/>.
00121 /// </summary>
00122     public class ActionDataPointElement :  IDataPointElement
00123     {
00124 /// <summary>
00125 /// The <see cref="Action"/> used to draw the symbol.  This should take as arguments the <see
      cref="Graphics"/>
```

```
00126 /// object on which to draw the symbol, the <see cref="PlotElementPresentationAttributes"/> describing
      the
00127 /// appearance of the symbol, and a <see langword="string"/> representing a tag for the symbol.
00128 /// </summary>
00129         public Action<Graphics, PlotElementPresentationAttributes, string> PlotAction { get; set; }
00130
00131 /// <inheritdoc/>
00132         public void Plot(Graphics target, PlotElementPresentationAttributes presentationAttributes,
      string tag) => PlotAction(target, presentationAttributes, tag);
00133
00134 /// <summary>
00135 /// Creates a new <see cref="ActionDataPointElement"/> using the specified action to draw the symbol.
00136 /// </summary>
00137 /// <param name="plotAction">The <see cref="Action"/> used to draw the symbol.  This should take as
      arguments the <see cref="Graphics"/>
00138 /// object on which to draw the symbol, the <see cref="PlotElementPresentationAttributes"/> describing
      the
00139 /// appearance of the symbol, and a <see langword="string"/> representing a tag for the
      symbol.</param>
00140         public ActionDataPointElement(Action<Graphics, PlotElementPresentationAttributes, string>
      plotAction)
00141         {
00142             PlotAction = plotAction;
00143         }
00144     }
00145
00146 /// <summary>
00147 /// A plot element that draws a symbol at the location of multiple data points.
00148 /// </summary>
00149 /// <typeparam name="T">The kind of data describing the data points (generally,
      <c>IReadOnlyList&lt;double&gt;</c>).</typeparam>
00150     public class ScatterPoints<T> :  IPlotElement
00151     {
00152 /// <summary>
00153 /// The data points at which the symbols will be drawn.
00154 /// </summary>
00155         public IEnumerable<T> Data { get; set; }
00156
00157 /// <summary>
00158 /// The size of the symbols to draw, in plot coordinates.
00159 /// </summary>
00160         public double Size { get; set; } = 2;
00161
00162 /// <summary>
00163 /// The symbol that will be drawn (by default, a circle).
00164 /// </summary>
00165         public IDataPointElement DataPointElement { get; set; } = new PathDataPointElement();
00166
00167 /// <summary>
00168 /// The coordinate system used to transform the points from data space to plot space.
00169 /// </summary>
00170         public ICoordinateSystem<T> CoordinateSystem { get; set; }
00171         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00172
00173 /// <summary>
00174 /// Presentation attributes determining the appearance (stroke and fill colour, etc.)  of the symbols.
00175 /// </summary>
00176         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes();
00177
00178 /// <summary>
00179 /// A tag to identify the symbols in the plot.
00180 /// </summary>
00181         public string Tag { get; set; }
00182
00183 /// <summary>
00184 /// Creates a new <see cref="ScatterPoints{T}"/> instance, using the specified <paramref name="data"/>
      and <paramref name="coordinateSystem"/>.
00185 /// </summary>
00186 /// <param name="data">The data points at which the symbols will be drawn.</param>
00187 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00188         public ScatterPoints(IEnumerable<T> data, ICoordinateSystem<T> coordinateSystem)
00189         {
00190             this.Data = data;
00191             this.CoordinateSystem = coordinateSystem;
00192         }
00193
00194 /// <inheritdoc/>
00195         public void Plot(Graphics target)
00196         {
00197             int index = 0;
00198
00199             foreach (T data in Data)
00200             {
00201                 Point pt = CoordinateSystem.ToPlotCoordinates(data);
00202
```

```
00203                 target.Save();
00204                 target.Translate(pt);
00205                 target.Scale(Size, Size);
00206 
00207                 if (!target.UseUniqueTags || string.IsNullOrEmpty(Tag))
00208                 {
00209                     DataPointElement.Plot(target, PresentationAttributes, Tag);
00210                 }
00211                 else
00212                 {
00213                     DataPointElement.Plot(target, PresentationAttributes, Tag + "@" +
      index.ToString());
00214                     index++;
00215                 }
00216 
00217                 target.Restore();
00218             }
00219         }
00220     }
00221 
00222 /// <summary>
00223 /// A plot element that draws a single text label.
00224 /// </summary>
00225 /// <typeparam name="T">The kind of data describing the data points (generally,
      <c>IReadOnlyList&lt;double&gt;</c>).</typeparam>
00226     public class TextLabel<T> :  IPlotElement
00227     {
00228 /// <summary>
00229 /// The position of the label, in data space coordinates.
00230 /// </summary>
00231         public T Position { get; set; }
00232 
00233 /// <summary>
00234 /// The text of the label.  This can include formatting specifiers (see the documentation for the <see
      cref="FormattedText.Format(string, Font, Font, Font, Font, Brush)"/> method).
00235 /// </summary>
00236         public string Label { get; set; }
00237 
00238 /// <summary>
00239 /// The angle at which the text is drawn, with respect to the horizontal.
00240 /// </summary>
00241         public double Rotation { get; set; } = 0;
00242 
00243 /// <summary>
00244 /// The baseline for the text.
00245 /// </summary>
00246         public TextBaselines Baseline { get; set; } = TextBaselines.Middle;
00247 
00248 /// <summary>
00249 /// The alignment for the text.
00250 /// </summary>
00251         public TextAnchors Alignment { get; set; } = TextAnchors.Center;
00252 
00253 /// <summary>
00254 /// The coordinate system used to transform the points from data space to plot space.
00255 /// </summary>
00256         public ICoordinateSystem<T> CoordinateSystem { get; set; }
00257         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00258 
00259 /// <summary>
00260 /// Presentation attributes determining the appearance (stroke and fill colour, etc.)  of the text
      label.
00261 /// </summary>
00262         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes() { Stroke = null };
00263 
00264 /// <summary>
00265 /// A tag to identify the label in the plot.
00266 /// </summary>
00267         public string Tag { get; set; }
00268 
00269 /// <summary>
00270 /// Create a new <see cref="TextLabel{T}"/> instance.
00271 /// </summary>
00272 /// <param name="label">The text of the label.  This can include formatting specifiers (see the
      documentation for the <see cref="FormattedText.Format(string, Font, Font, Font, Font, Brush)"/>
      method).</param>
00273 /// <param name="position">The position of the label, in data space coordinates.</param>
00274 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00275         public TextLabel(string label, T position, ICoordinateSystem<T> coordinateSystem)
00276         {
00277             this.Label = label;
00278             this.Position = position;
00279             this.CoordinateSystem = coordinateSystem;
00280         }
00281 
```

```
00282 /// <inheritdoc/>
00283         public void Plot(Graphics target)
00284         {
00285             if (!string.IsNullOrEmpty(Label))
00286             {
00287                 IEnumerable<FormattedText> label;
00288 
00289                 if (PresentationAttributes.Font.FontFamily.IsStandardFamily)
00290                 {
00291                     int i = Array.IndexOf(FontFamily.StandardFamilies,
      PresentationAttributes.Font.FontFamily.FamilyName.Replace(" ", "-"));
00292 
00293                     label = FormattedText.Format(this.Label, (FontFamily.StandardFontFamilies)i,
      PresentationAttributes.Font.FontSize);
00294                 }
00295                 else
00296                 {
00297                     label = FormattedText.Format(this.Label, PresentationAttributes.Font,
      PresentationAttributes.Font, PresentationAttributes.Font, PresentationAttributes.Font);
00298                 }
00299 
00300                 Point point = CoordinateSystem.ToPlotCoordinates(Position);
00301 
00302                 target.Save();
00303                 target.Translate(point);
00304                 target.Rotate(Rotation);
00305 
00306                 double x = Alignment == TextAnchors.Left ?  0 :  Alignment == TextAnchors.Right ?
      -label.Measure().Width :  -label.Measure().Width * 0.5;
00307 
00308                 string fillTag = Tag;
00309                 string strokeTag = Tag;
00310 
00311                 if (!string.IsNullOrEmpty(Tag) && target.UseUniqueTags)
00312                 {
00313                     strokeTag = strokeTag + "@stroke";
00314                 }
00315 
00316                 if (PresentationAttributes.Fill != null)
00317                 {
00318                     target.FillText(x, 0, label, PresentationAttributes.Fill, Baseline, fillTag);
00319                 }
00320 
00321                 if (PresentationAttributes.Stroke != null)
00322                 {
00323                     target.StrokeText(x, 0, label, PresentationAttributes.Stroke, Baseline,
      PresentationAttributes.LineWidth, PresentationAttributes.LineCap, PresentationAttributes.LineJoin,
      PresentationAttributes.LineDash, strokeTag);
00324                 }
00325 
00326                 target.Restore();
00327             }
00328         }
00329 
00330     }
00331 
00332 /// <summary>
00333 /// A plot element that draws a text label at each data point.
00334 /// </summary>
00335 /// <typeparam name="T">The kind of data describing the data points (generally,
      <c>IReadOnlyList&lt;double&gt;</c>).</typeparam>
00336     public class DataLabels<T> :  IPlotElement
00337     {
00338 /// <summary>
00339 /// The data points at which the labels will be drawn.
00340 /// </summary>
00341         public IEnumerable<T> Data { get; set; }
00342 
00343         private Func<int, T, IEnumerable<FormattedText» labelFunction;
00344         private Func<int, T, object> label;
00345 
00346 /// <summary>
00347 /// A function used to determine the text of the labels to draw.  The arguments for this function
      should be a <see langword="int"/>
00348 /// representing the index of the data point and a <typeparamref name="T"/> representing the
      coordinates of the data point.  This
00349 /// function should return either an <see cref="IEnumerable{T}"/> of <see cref="FormattedText"/>
      objects, which will be used as-is,
00350 /// or any other kind of object, which will be converted to a <see langword="string"/> to be used in
      the label.
00351 /// </summary>
00352         public Func<int, T, object> Label
00353         {
00354             get
00355             {
00356                 return label;
00357             }
```

```
00358
00359             set
00360             {
00361                 label = value;
00362
00363                 if (typeof(IEnumerable<FormattedText>).IsAssignableFrom(value.Method.ReturnType))
00364                 {
00365                     labelFunction = (i, a) => (IEnumerable<FormattedText>)value(i, a);
00366                 }
00367                 else
00368                 {
00369                     labelFunction = (i, a) =>
00370                     {
00371                         object lbl = value(i, a);
00372
00373                         if (lbl is IEnumerable<FormattedText> frmt)
00374                         {
00375                             return frmt;
00376                         }
00377                         else if (lbl == null)
00378                         {
00379                             return null;
00380                         }
00381                         else
00382                         {
00383                             if (PresentationAttributes.Font.FontFamily.IsStandardFamily)
00384                             {
00385                                 int j = Array.IndexOf(FontFamily.StandardFamilies,
    PresentationAttributes.Font.FontFamily.FamilyName.Replace(" ", "-"));
00386
00387                                 return FormattedText.Format(lbl.ToString(),
    (FontFamily.StandardFontFamilies)j, PresentationAttributes.Font.FontSize);
00388                             }
00389                             else
00390                             {
00391                                 return FormattedText.Format(lbl.ToString(),
    PresentationAttributes.Font, PresentationAttributes.Font, PresentationAttributes.Font,
    PresentationAttributes.Font);
00392                             }
00393                         }
00394                     };
00395                 }
00396             }
00397         }
00398
00399 /// <summary>
00400 /// A function used to determine the orientation of the labels with respect to the horizontal.  The
    arguments for this function should be a <see langword="int"/>
00401 /// representing the index of the data point and a <typeparamref name="T"/> representing the
    coordinates of the data point.  This
00402 /// function should return a <see langword="double"/> representing the angle with respect to the
    horizontal at which the label
00403 /// should be drawn.
00404 /// </summary>
00405         public Func<int, T, double> Rotation { get; set; } = (i, d) => 0;
00406
00407 /// <summary>
00408 /// A function used to determine the position of the labels with respect to the data points.  The
    arguments for this function should be a <see langword="int"/>
00409 /// representing the index of the data point and a <typeparamref name="T"/> representing the
    coordinates of the data point.  This
00410 /// function should return a <see cref="Point"/> defining the amount of space between the data point
    and the label, in plot coordinates.
00411 /// </summary>
00412         public Func<int, T, Point> Margin { get; set; } = (i, d) => new Point();
00413
00414 /// <summary>
00415 /// The baseline for the labels.
00416 /// </summary>
00417         public TextBaselines Baseline { get; set; } = TextBaselines.Middle;
00418
00419 /// <summary>
00420 /// The alignment for the labels.
00421 /// </summary>
00422         public TextAnchors Alignment { get; set; } = TextAnchors.Center;
00423
00424 /// <summary>
00425 /// The coordinate system used to transform the points from data space to plot space.
00426 /// </summary>
00427         public ICoordinateSystem<T> CoordinateSystem { get; set; }
00428         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00429
00430 /// <summary>
00431 /// Presentation attributes determining the appearance of the labels.
00432 /// </summary>
00433         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
    PlotElementPresentationAttributes() { Stroke = null };
```

```
00434
00435 /// <summary>
00436 /// A tag to identify the labels in the plot.
00437 /// </summary>
00438         public string Tag { get; set; }
00439
00440 /// <summary>
00441 /// Creates a new <see cref="DataLabels{T}"/> instance.
00442 /// </summary>
00443 /// <param name="data">The data points at which the labels will be drawn.</param>
00444 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00445         public DataLabels(IEnumerable<T> data, ICoordinateSystem<T> coordinateSystem)
00446         {
00447             this.Data = data;
00448             this.CoordinateSystem = coordinateSystem;
00449             this.Label = (i, d) =>
00450             {
00451                 if (PresentationAttributes.Font.FontFamily.IsStandardFamily)
00452                 {
00453                     int j = Array.IndexOf(FontFamily.StandardFamilies,
      PresentationAttributes.Font.FontFamily.FamilyName.Replace(" ", "-"));
00454
00455                     return FormattedText.Format(i.ToString(), (FontFamily.StandardFontFamilies)j,
      PresentationAttributes.Font.FontSize);
00456                 }
00457                 else
00458                 {
00459                     return FormattedText.Format(i.ToString(), PresentationAttributes.Font,
      PresentationAttributes.Font, PresentationAttributes.Font, PresentationAttributes.Font);
00460                 }
00461             };
00462         }
00463
00464 /// <inheritdoc/>
00465         public void Plot(Graphics target)
00466         {
00467             int index = 0;
00468
00469             foreach (T data in Data)
00470             {
00471                 IEnumerable<FormattedText> label = labelFunction(index, data);
00472                 if (label != null)
00473                 {
00474                     double rotation = Rotation(index, data);
00475                     Point pt = CoordinateSystem.ToPlotCoordinates(data);
00476                     Point margin = Margin(index, data);
00477
00478                     pt = new Point(pt.X + margin.X, pt.Y + margin.Y);
00479
00480                     target.Save();
00481                     target.Translate(pt);
00482                     target.Rotate(rotation);
00483
00484                     string fillTag = Tag;
00485                     string strokeTag = Tag;
00486
00487
00488                     if (target.UseUniqueTags && !string.IsNullOrEmpty(Tag))
00489                     {
00490                         fillTag = fillTag + "@" + index.ToString();
00491                         strokeTag = strokeTag + "@stroke" + index.ToString();
00492                     }
00493
00494                     if (PresentationAttributes.Fill != null)
00495                     {
00496                         target.FillText(Alignment == TextAnchors.Left ?  0 :  Alignment ==
      TextAnchors.Right ?  -label.Measure().Width :  -label.Measure().Width * 0.5, 0, label,
      PresentationAttributes.Fill, Baseline, fillTag);
00497                     }
00498
00499                     if (PresentationAttributes.Stroke != null)
00500                     {
00501                         target.StrokeText(Alignment == TextAnchors.Left ?  0 :  Alignment ==
      TextAnchors.Right ?  -label.Measure().Width :  -label.Measure().Width * 0.5, 0, label,
      PresentationAttributes.Stroke, Baseline, PresentationAttributes.LineWidth,
      PresentationAttributes.LineCap, PresentationAttributes.LineJoin, PresentationAttributes.LineDash,
      fillTag);
00502                     }
00503
00504                     target.Restore();
00505                 }
00506                 index++;
00507             }
00508         }
00509     }
00510
```

```
00511 /// <summary>
00512 /// A plot element that draws a line passing through a set of points.
00513 /// </summary>
00514 /// <typeparam name="T">The kind of data describing the data points (generally,
      <c>IReadOnlyList&lt;double&gt;</c>).</typeparam>
00515     public class DataLine<T> :  IPlotElement
00516     {
00517 /// <summary>
00518 /// The data points through which the line will pass.
00519 /// </summary>
00520         public IEnumerable<T> Data { get; set; }
00521
00522 /// <summary>
00523 /// If this is <see langword="false"/>, straight line segments are used to join the data points.  If
      this is <see langword="true"/>,
00524 /// a smooth spline passing through all of them is used instead.
00525 /// </summary>
00526         public bool Smooth { get; set; }
00527
00528 /// <summary>
00529 /// The coordinate system used to transform the points from data space to plot space.
00530 /// </summary>
00531         public ICoordinateSystem<T> CoordinateSystem { get; set; }
00532         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00533
00534 /// <summary>
00535 /// Presentation attributes determining the appearance of the line.
00536 /// </summary>
00537         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes();
00538
00539 /// <summary>
00540 /// A tag to identify the labels in the plot.
00541 /// </summary>
00542         public string Tag { get; set; }
00543
00544 /// <summary>
00545 /// Create a new instance of the <see cref="DataLine{T}"/> class.
00546 /// </summary>
00547 /// <param name="data">The data points through which the line will pass.</param>
00548 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00549         public DataLine(IEnumerable<T> data, ICoordinateSystem<T> coordinateSystem)
00550         {
00551             this.Data = data;
00552             this.CoordinateSystem = coordinateSystem;
00553         }
00554
00555 /// <inheritdoc/>
00556         public void Plot(Graphics target)
00557         {
00558             GraphicsPath pth = new GraphicsPath();
00559
00560             if (!Smooth)
00561             {
00562                 bool started = false;
00563
00564                 foreach (T data in Data)
00565                 {
00566                     Point pt = CoordinateSystem.ToPlotCoordinates(data);
00567
00568                     if (!started)
00569                     {
00570                         pth.MoveTo(pt);
00571                         started = true;
00572                     }
00573                     else
00574                     {
00575                         pth.LineTo(pt);
00576                     }
00577                 }
00578             }
00579             else
00580             {
00581                 List<Point> points = new List<Point>();
00582
00583                 foreach (T data in Data)
00584                 {
00585                     Point pt = CoordinateSystem.ToPlotCoordinates(data);
00586                     points.Add(pt);
00587                 }
00588                 pth.AddSmoothSpline(points.ToArray());
00589             }
00590
00591             target.StrokePath(pth, PresentationAttributes.Stroke, PresentationAttributes.LineWidth,
      PresentationAttributes.LineCap, PresentationAttributes.LineJoin, PresentationAttributes.LineDash,
      Tag);
```

```
00592         }
00593     }
00594
00595 /// <summary>
00596 /// A plot element that fills an area between a line passing through some data points and a base line.
00597 /// </summary>
00598 /// <typeparam name="T">The kind of data describing the data points (generally,
      <c>IReadOnlyList&lt;double&gt;</c>).</typeparam>
00599     public class Area<T> :  IPlotElement
00600     {
00601 /// <summary>
00602 /// The data points through which the upper part of the area will pass.
00603 /// </summary>
00604         public IEnumerable<T> Data { get; set; }
00605
00606 /// <summary>
00607 /// A function returning the baseline for each data point.
00608 /// </summary>
00609         public Func<T, T> GetBaseline { get; set; }
00610
00611 /// <summary>
00612 /// If this is <see langword="false"/>, straight line segments are used to join the data points.  If
      this is <see langword="true"/>,
00613 /// a smooth spline passing through all of them is used instead.
00614 /// </summary>
00615         public bool Smooth { get; set; }
00616
00617
00618 /// <summary>
00619 /// The coordinate system used to transform the points from data space to plot space.
00620 /// </summary>
00621         public ICoordinateSystem<T> CoordinateSystem { get; set; }
00622         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00623
00624 /// <summary>
00625 /// Presentation attributes determining the appearance of the area.
00626 /// </summary>
00627         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes();
00628
00629 /// <summary>
00630 /// A tag to identify the area in the plot.
00631 /// </summary>
00632         public string Tag { get; set; }
00633
00634 /// <summary>
00635 /// Create a new <see cref="Area{T}"/> instance.
00636 /// </summary>
00637 /// <param name="data">The data points through which the upper part of the area will pass.</param>
00638 /// <param name="getBaseline">A function returning the baseline for each data point.</param>
00639 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00640         public Area(IEnumerable<T> data, Func<T, T> getBaseline, ICoordinateSystem<T>
      coordinateSystem)
00641         {
00642             this.Data = data;
00643             this.CoordinateSystem = coordinateSystem;
00644             this.GetBaseline = getBaseline;
00645         }
00646
00647 /// <inheritdoc/>
00648         public void Plot(Graphics target)
00649         {
00650             GraphicsPath pth = new GraphicsPath();
00651             GraphicsPath pthStroke = new GraphicsPath();
00652
00653             if (!Smooth)
00654             {
00655                 bool started = false;
00656
00657                 List<Point> oppositePoints = new List<Point>();
00658
00659                 foreach (T data in Data)
00660                 {
00661                     Point pt = CoordinateSystem.ToPlotCoordinates(data);
00662
00663                     oppositePoints.Add(CoordinateSystem.ToPlotCoordinates(GetBaseline(data)));
00664
00665                     if (!started)
00666                     {
00667                         pth.MoveTo(pt);
00668                         pthStroke.MoveTo(pt);
00669                         started = true;
00670                     }
00671                     else
00672                     {
00673                         pth.LineTo(pt);
```

```
00674                         pthStroke.LineTo(pt);
00675                     }
00676                 }
00677 
00678                 for (int i = oppositePoints.Count - 1; i >= 0; i--)
00679                 {
00680                     pth.LineTo(oppositePoints[i]);
00681                 }
00682 
00683                 pth.Close();
00684             }
00685             else
00686             {
00687                 List<Point> points = new List<Point>();
00688                 List<Point> oppositePoints = new List<Point>();
00689 
00690                 foreach (T data in Data)
00691                 {
00692                     Point pt = CoordinateSystem.ToPlotCoordinates(data);
00693                     points.Add(pt);
00694                     oppositePoints.Add(CoordinateSystem.ToPlotCoordinates(GetBaseline(data)));
00695                 }
00696                 pth.AddSmoothSpline(points.ToArray());
00697                 pthStroke.AddSmoothSpline(points.ToArray());
00698                 oppositePoints.Reverse();
00699                 pth.AddSmoothSpline(oppositePoints.ToArray());
00700                 pth.Close();
00701             }
00702 
00703             string tag = Tag;
00704             string strokeTag = tag;
00705 
00706             if (!string.IsNullOrEmpty(tag) && target.UseUniqueTags)
00707             {
00708                 strokeTag += "@stroke";
00709             }
00710 
00711             if (PresentationAttributes.Fill != null)
00712             {
00713                 target.FillPath(pth, PresentationAttributes.Fill, tag);
00714             }
00715 
00716             if (PresentationAttributes.Stroke != null)
00717             {
00718                 target.StrokePath(pthStroke, PresentationAttributes.Stroke,
      PresentationAttributes.LineWidth, PresentationAttributes.LineCap, PresentationAttributes.LineJoin,
      PresentationAttributes.LineDash, strokeTag);
00719             }
00720         }
00721     }
00722 }
```

8.41 Function2D.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using MathNet.Numerics.Distributions;
00019 using System;
00020 using System.Collections.Generic;
00021 using System.Linq;
00022 using System.Runtime.InteropServices;
00023 
00024 namespace VectSharp.Plots
00025 {
00026 /// <summary>
00027 /// Represents a function of two variables that has been sampled in some points.
00028 /// </summary>
00029     public class Function2DGrid
```

```
00030     {
00031 /// <summary>
00032 /// Describes the arrangement of the points that have been sampled.
00033 /// </summary>
00034         public enum GridType
00035         {
00036 /// <summary>
00037 /// Points have been sampled along a rectangular grid.
00038 /// </summary>
00039             Rectangular,
00040 
00041 /// <summary>
00042 /// Points have been sampled along a grid composed of hexagons with a diagonal
00043 /// parallel to the horizontal.
00044 /// </summary>
00045             HexagonHorizontal,
00046 
00047 /// <summary>
00048 /// Points have been sampled along a grid composed of hexagons with a diagonal
00049 /// parallel to the vertical.
00050 /// </summary>
00051             HexagonVertical,
00052 
00053 /// <summary>
00054 /// Points have been sampled without any particular criterion.
00055 /// </summary>
00056             Irregular
00057         }
00058 
00059 /// <summary>
00060 /// The points where the function has been sampled.
00061 /// </summary>
00062         public IReadOnlyList<IReadOnlyList<double» DataPoints { get; }
00063 
00064 /// <summary>
00065 /// The minimum X value that has been sampled.
00066 /// </summary>
00067         public double MinX { get; }
00068 
00069 /// <summary>
00070 /// The maximum X value that has been sampled.
00071 /// </summary>
00072         public double MaxX { get; }
00073 
00074 /// <summary>
00075 /// The minimum Y value that has been sampled.
00076 /// </summary>
00077         public double MinY { get; }
00078 
00079 /// <summary>
00080 /// The maximum Y value that has been sampled.
00081 /// </summary>
00082         public double MaxY { get; }
00083 
00084 /// <summary>
00085 /// The minimum value that has been obtained for the function.
00086 /// </summary>
00087         public double MinZ { get; }
00088 
00089 /// <summary>
00090 /// The maximum value that has been obtained for the function.
00091 /// </summary>
00092         public double MaxZ { get; }
00093 
00094 /// <summary>
00095 /// If the function has been sampled along a regular grid, the number of steps
00096 /// between <see cref="MinX"/> and <see cref="MaxX"/> on the X axis.
00097 /// </summary>
00098         public int StepsX { get; private set; }
00099 
00100 /// <summary>
00101 /// If the function has been sampled along a regular grid, the number of steps
00102 /// between <see cref="MinY"/> and <see cref="MaxY"/> on the X axis.
00103 /// </summary>
00104         public int StepsY { get; private set; }
00105 
00106 /// <summary>
00107 /// The type of grid.
00108 /// </summary>
00109         public GridType Type { get; private set; }
00110 
00111 /// <summary>
00112 /// Create a new <see cref="Function2DGrid"/> from a list of sampled values.
00113 /// </summary>
00114 /// <param name="dataPoints">The sampled values.  Each element of this <see cref="IReadOnlyList{T}"/>
     should
00115 /// be a collection of three values:  the X coordinate of the sampled point, the Y coordinate of the
```

```
      sampled
00116 /// point, and the value of the function at that point.</param>
00117         public Function2DGrid(IReadOnlyList<IReadOnlyList<double» dataPoints)
00118         {
00119             this.Type = GridType.Irregular;
00120             this.DataPoints = dataPoints.ToArray();
00121 
00122             MinX = double.MaxValue;
00123             MaxX = double.MinValue;
00124             MinY = double.MaxValue;
00125             MaxY = double.MinValue;
00126             MinZ = double.MaxValue;
00127             MaxZ = double.MinValue;
00128 
00129             this.StepsX = 0;
00130             this.StepsY = 0;
00131 
00132             for (int i = 0; i < dataPoints.Count; i++)
00133             {
00134                 MinX = Math.Min(MinX, dataPoints[i][0]);
00135                 MinY = Math.Min(MinY, dataPoints[i][1]);
00136                 MinZ = Math.Min(MinZ, dataPoints[i][2]);
00137 
00138                 MaxX = Math.Max(MaxX, dataPoints[i][0]);
00139                 MaxY = Math.Max(MaxY, dataPoints[i][1]);
00140                 MaxZ = Math.Max(MaxZ, dataPoints[i][2]);
00141             }
00142         }
00143 
00144 /// <summary>
00145 /// Create a new <see cref="Function2DGrid"/> by sampling the provided function.
00146 /// </summary>
00147 /// <param name="function">The function to sample.</param>
00148 /// <param name="minX">The minimum X value at which the function should be sampled.</param>
00149 /// <param name="minY">The minimum Y value at which the function should be sampled.</param>
00150 /// <param name="maxX">The maximum X value at which the function should be sampled.</param>
00151 /// <param name="maxY">The maximum Y value at which the function should be sampled.</param>
00152 /// <param name="stepsX">If <paramref name="type"/> is not <see cref="GridType.Irregular"/>,
00153 /// the number of steps between <paramref name="minX"/> and <paramref name="maxX"/> on the X axis.
00154 /// Otherwise, the number of sampled points is determined by multiplying <paramref name="stepsX"/> and
00155 /// <paramref name="stepsY"/> together.</param>
00156 /// <param name="stepsY">If <paramref name="type"/> is not <see cref="GridType.Irregular"/>,
00157 /// the number of steps between <paramref name="minY"/> and <paramref name="maxY"/> on the Y axis.
00158 /// Otherwise, the number of sampled points is determined by multiplying <paramref name="stepsX"/> and
00159 /// <paramref name="stepsY"/> together.</param>
00160 /// <param name="type">The strategy used to select points to sample.  If this is <see
      cref="GridType.Irregular"/>,
00161 /// uniformly distributed random points between (<paramref name="minX"/>, <paramref name="minY"/>) and
00162 /// (<paramref name="maxX"/>, <paramref name="maxY"/>) are sampled.</param>
00163         public Function2DGrid(Func<double[], double> function, double minX, double minY, double maxX,
      double maxY, int stepsX, int stepsY, GridType type)
00164         {
00165             MinX = minX;
00166             MaxX = maxX;
00167             MinY = minY;
00168             MaxY = maxY;
00169             StepsX = stepsX;
00170             StepsY = stepsY;
00171             Type = type;
00172             MinZ = double.MaxValue;
00173             MaxZ = double.MinValue;
00174 
00175             if (type == GridType.HexagonHorizontal)
00176             {
00177                 List<double[]> dataPoints = new List<double[]>((stepsX + 1) * (stepsY + 1));
00178 
00179                 for (int y = 0; y <= stepsY; y++)
00180                 {
00181                     for (int x = 0; x <= stepsX; x++)
00182                     {
00183                         if (x % 2 == 0 || y < stepsY)
00184                         {
00185 
00186                             double realY;
00187 
00188                             if (x % 2 == 0)
00189                             {
00190                                 realY = minY + (maxY - minY) / stepsY * y;
00191                             }
00192                             else
00193                             {
00194                                 realY = minY + (maxY - minY) / stepsY * (y + 0.5);
00195                             }
00196 
00197                             double realX = minX + (maxX - minX) / stepsX * x;
00198                             double z = function(new double[] { realX, realY });
00199 
```

```
00200                           if (!double.IsNaN(z))
00201                           {
00202                               MinZ = Math.Min(MinZ, z);
00203                               MaxZ = Math.Max(MaxZ, z);
00204 
00205                               dataPoints.Add(new double[] { realX, realY, z });
00206                           }
00207                           else
00208                           {
00209                               Type = GridType.Irregular;
00210                           }
00211                       }
00212                   }
00213               }
00214 
00215               this.DataPoints = dataPoints;
00216           }
00217           else if (type == GridType.HexagonVertical)
00218           {
00219               List<double[]> dataPoints = new List<double[]>((stepsX + 1) * (stepsY + 1));
00220 
00221               for (int y = 0; y <= stepsY; y++)
00222               {
00223                   double realY = minY + (maxY - minY) / stepsY * y;
00224 
00225                   for (int x = 0; x <= stepsX; x++)
00226                   {
00227                       if (y % 2 == 0 || x < stepsX)
00228                       {
00229 
00230                           double realX;
00231 
00232                           if (y % 2 == 0)
00233                           {
00234                               realX = minX + (maxX - minX) / stepsX * x;
00235                           }
00236                           else
00237                           {
00238                               realX = minX + (maxX - minX) / stepsX * (x + 0.5);
00239                           }
00240 
00241                           double z = function(new double[] { realX, realY });
00242 
00243                           if (!double.IsNaN(z))
00244                           {
00245                               MinZ = Math.Min(MinZ, z);
00246                               MaxZ = Math.Max(MaxZ, z);
00247 
00248                               dataPoints.Add(new double[] { realX, realY, z });
00249                           }
00250                           else
00251                           {
00252                               Type = GridType.Irregular;
00253                           }
00254                       }
00255                   }
00256               }
00257 
00258               this.DataPoints = dataPoints;
00259           }
00260           else if (type == GridType.Rectangular)
00261           {
00262               List<double[]> dataPoints = new List<double[]>((stepsX + 1) * (stepsY + 1));
00263 
00264               for (int y = 0; y <= stepsY; y++)
00265               {
00266                   double realY = minY + (maxY - minY) / stepsY * y;
00267                   for (int x = 0; x <= stepsX; x++)
00268                   {
00269                       double realX = minX + (maxX - minX) / stepsX * x;
00270                       double z = function(new double[] { realX, realY });
00271 
00272                       if (!double.IsNaN(z))
00273                       {
00274                           MinZ = Math.Min(MinZ, z);
00275                           MaxZ = Math.Max(MaxZ, z);
00276 
00277                           dataPoints.Add(new double[] { realX, realY, z });
00278                       }
00279                       else
00280                       {
00281                           Type = GridType.Irregular;
00282                       }
00283                   }
00284               }
00285 
00286               this.DataPoints = dataPoints;
```

```
00287             }
00288             else //type == GridType.Irregular
00289             {
00290                 double[] xs = ContinuousUniform.Samples(minX, maxX).Take((stepsX + 1) * (stepsY +
    1)).ToArray();
00291                 double[] ys = ContinuousUniform.Samples(minY, maxY).Take((stepsX + 1) * (stepsY +
    1)).ToArray();
00292 
00293                 List<double[]> dataPoints = new List<double[]>((stepsX + 1) * (stepsY + 1));
00294 
00295                 for (int i = 0; i < (stepsX + 1) * (stepsY + 1); i++)
00296                 {
00297                     double z = function(new double[] { xs[i], ys[i] });
00298 
00299                     if (!double.IsNaN(z))
00300                     {
00301                         MinZ = Math.Min(MinZ, z);
00302                         MaxZ = Math.Max(MaxZ, z);
00303 
00304                         dataPoints.Add(new double[] { xs[i], ys[i], z });
00305                     }
00306                 }
00307 
00308                 this.DataPoints = dataPoints;
00309             }
00310         }
00311 
00312 /// <summary>
00313 /// Converts a hexagonal grid into a rectangular grid.
00314 /// </summary>
00315 /// <returns>If <see cref="Type"/> is <see cref="GridType.Rectangular"/>, this method returns the
    current instance.  <br/>
00316 /// If <see cref="Type"/> is <see cref="GridType.HexagonHorizontal"/> or <see
    cref="GridType.HexagonVertical"/>, a
00317 /// new <see cref="Function2DGrid"/> with <see cref="Type"/> equal to <see
    cref="GridType.Rectangular"/> is returned.
00318 /// The sampled points in this grid are obtained by performing a bilinear interpolation on the sampled
    points from
00319 /// this grid.  The returned grid will always be "denser" than the current instance.<br/>
00320 /// If <see cref="Type"/> is <see cref="GridType.Irregular"/>, an <see
    cref="InvalidOperationException"/> is thrown.</returns>
00321 /// <exception cref="InvalidOperationException">Thrown if <see cref="Type"/> is <see
    cref="GridType.Irregular"/>.</exception>
00322         public Function2DGrid ToRectangular()
00323         {
00324             if (this.Type == GridType.Rectangular)
00325             {
00326                 return this;
00327             }
00328             else if (this.Type == GridType.HexagonHorizontal)
00329             {
00330                 IReadOnlyList<double>[] data = new IReadOnlyList<double>[(2 * this.StepsY + 1) *
    (this.StepsX + 1)];
00331 
00332                 int newStepsX = this.StepsX;
00333                 int newStepsY = this.StepsY * 2;
00334 
00335                 for (int i = 0; i < this.DataPoints.Count; i++)
00336                 {
00337                     int x = (int)Math.Round((this.DataPoints[i][0] - this.MinX) / (this.MaxX -
    this.MinX) * newStepsX);
00338                     int y = (int)Math.Round((this.DataPoints[i][1] - this.MinY) / (this.MaxY -
    this.MinY) * newStepsY);
00339 
00340                     data[y * (newStepsX + 1) + x] = this.DataPoints[i];
00341                 }
00342 
00343                 for (int y = 0; y <= newStepsY; y++)
00344                 {
00345                     for (int x = 0; x <= newStepsX; x++)
00346                     {
00347                         if (data[y * (newStepsX + 1) + x] == null)
00348                         {
00349                             double xVal = this.MinX + (this.MaxX - this.MinX) / newStepsX * x;
00350                             double yVal = this.MinY + (this.MaxY - this.MinY) / newStepsY * y;
00351 
00352                             double zVal;
00353 
00354                             if (x > 0)
00355                             {
00356                                 if (y > 0)
00357                                 {
00358                                     if (x < newStepsX)
00359                                     {
00360                                         if (y < newStepsY)
00361                                         {
00362                                             zVal = (data[(y - 1) * (newStepsX + 1) + x][2] + data[(y +
```

```
1) * (newStepsX + 1) + x][2] + data[y * (newStepsX + 1) + (x - 1)][2] + data[y * (newStepsX + 1) + (x
+ 1)][2]) * 0.25;
00363                                       }
00364                                       else
00365                                       {
00366                                           zVal = (data[y * (newStepsX + 1) + (x - 1)][2] + data[y *
(newStepsX + 1) + (x + 1)][2]) * 0.5;
00367                                       }
00368                                   }
00369                                   else
00370                                   {
00371                                       if (y < newStepsY)
00372                                       {
00373                                           zVal = (data[(y - 1) * (newStepsX + 1) + x][2] + data[(y +
1) * (newStepsX + 1) + x][2]) * 0.5;
00374                                       }
00375                                       else
00376                                       {
00377                                           zVal = (data[y * (newStepsX + 1) + (x - 1)][2] + data[(y -
1) * (newStepsX + 1) + x][2]) * 0.5;
00378                                       }
00379                                   }
00380                               }
00381                               else
00382                               {
00383                                   if (x < newStepsX)
00384                                   {
00385                                       zVal = (data[y * (newStepsX + 1) + (x - 1)][2] + data[y *
(newStepsX + 1) + (x + 1)][2]) * 0.5;
00386                                   }
00387                                   else
00388                                   {
00389                                       zVal = (data[y * (newStepsX + 1) + (x - 1)][2] + data[(y + 1)
* (newStepsX + 1) + x][2]) * 0.5;
00390                                   }
00391                               }
00392                           }
00393                           else
00394                           {
00395                               if (y > 0)
00396                               {
00397                                   if (y < newStepsY)
00398                                   {
00399                                       zVal = (data[(y - 1) * (newStepsX + 1) + x][2] + data[(y + 1)
* (newStepsX + 1) + x][2]) * 0.5;
00400                                   }
00401                                   else
00402                                   {
00403                                       zVal = (data[y * (newStepsX + 1) + (x + 1)][2] + data[(y - 1)
* (newStepsX + 1) + x][2]) * 0.5;
00404                                   }
00405                               }
00406                               else
00407                               {
00408                                   zVal = (data[y * (newStepsX + 1) + (x + 1)][2] + data[(y + 1) *
(newStepsX + 1) + x][2]) * 0.5;
00409                               }
00410                           }
00411
00412                           data[y * (newStepsX + 1) + x] = new double[] { xVal, yVal, zVal };
00413                       }
00414                   }
00415               }
00416
00417               return new Function2DGrid(data) { Type = GridType.Rectangular, StepsX = newStepsX,
StepsY = newStepsY };
00418           }
00419           else if (this.Type == GridType.HexagonVertical)
00420           {
00421               IReadOnlyList<double>[] data = new IReadOnlyList<double>[(this.StepsY + 1) * (2 *
this.StepsX + 1)];
00422
00423               int newStepsX = this.StepsX * 2;
00424               int newStepsY = this.StepsY;
00425
00426               for (int i = 0; i < this.DataPoints.Count; i++)
00427               {
00428                   int x = (int)Math.Round((this.DataPoints[i][0] - this.MinX) / (this.MaxX -
this.MinX) * newStepsX);
00429                   int y = (int)Math.Round((this.DataPoints[i][1] - this.MinY) / (this.MaxY -
this.MinY) * newStepsY);
00430
00431                   data[y * (newStepsX + 1) + x] = this.DataPoints[i];
00432               }
00433
00434               for (int y = 0; y <= newStepsY; y++)
00435               {
```

```
00436                     for (int x = 0; x <= newStepsX; x++)
00437                     {
00438                         if (data[y * (newStepsX + 1) + x] == null)
00439                         {
00440                             double xVal = this.MinX + (this.MaxX - this.MinX) / newStepsX * x;
00441                             double yVal = this.MinY + (this.MaxY - this.MinY) / newStepsY * y;
00442 
00443                             double zVal;
00444 
00445                             if (x > 0)
00446                             {
00447                                 if (y > 0)
00448                                 {
00449                                     if (x < newStepsX)
00450                                     {
00451                                         if (y < newStepsY)
00452                                         {
00453                                             zVal = (data[(y - 1) * (newStepsX + 1) + x][2] + data[(y + 1) * (newStepsX + 1) + x][2] + data[y * (newStepsX + 1) + (x - 1)][2] + data[y * (newStepsX + 1) + (x + 1)][2]) * 0.25;
00454                                         }
00455                                         else
00456                                         {
00457                                             zVal = (data[y * (newStepsX + 1) + (x - 1)][2] + data[y * (newStepsX + 1) + (x + 1)][2]) * 0.5;
00458                                         }
00459                                     }
00460                                     else
00461                                     {
00462                                         if (y < newStepsY)
00463                                         {
00464                                             zVal = (data[(y - 1) * (newStepsX + 1) + x][2] + data[(y + 1) * (newStepsX + 1) + x][2]) * 0.5;
00465                                         }
00466                                         else
00467                                         {
00468                                             zVal = (data[y * (newStepsX + 1) + (x - 1)][2] + data[(y - 1) * (newStepsX + 1) + x][2]) * 0.5;
00469                                         }
00470                                     }
00471                                 }
00472                                 else
00473                                 {
00474                                     if (x < newStepsX)
00475                                     {
00476                                         zVal = (data[y * (newStepsX + 1) + (x - 1)][2] + data[y * (newStepsX + 1) + (x + 1)][2]) * 0.5;
00477                                     }
00478                                     else
00479                                     {
00480                                         zVal = (data[y * (newStepsX + 1) + (x - 1)][2] + data[(y + 1) * (newStepsX + 1) + x][2]) * 0.5;
00481                                     }
00482                                 }
00483                             }
00484                             else
00485                             {
00486                                 if (y > 0)
00487                                 {
00488                                     if (y < newStepsY)
00489                                     {
00490                                         zVal = (data[(y - 1) * (newStepsX + 1) + x][2] + data[(y + 1) * (newStepsX + 1) + x][2]) * 0.5;
00491                                     }
00492                                     else
00493                                     {
00494                                         zVal = (data[y * (newStepsX + 1) + (x + 1)][2] + data[(y - 1) * (newStepsX + 1) + x][2]) * 0.5;
00495                                     }
00496                                 }
00497                                 else
00498                                 {
00499                                     zVal = (data[y * (newStepsX + 1) + (x + 1)][2] + data[(y + 1) * (newStepsX + 1) + x][2]) * 0.5;
00500                                 }
00501                             }
00502 
00503                             data[y * (newStepsX + 1) + x] = new double[] { xVal, yVal, zVal };
00504                         }
00505                     }
00506                 }
00507 
00508                 return new Function2DGrid(data) { Type = GridType.Rectangular, StepsX = newStepsX, StepsY = newStepsY };
00509             }
00510             else
00511             {
```

```
00512                 throw new InvalidOperationException("Cannot convert an irregular grid into a
      rectangular grid!");
00513             }
00514         }
00515     }
00516
00517 /// <summary>
00518 /// A plot element that plots a function of two variables.
00519 /// </summary>
00520     public class Function2D :  IPlotElement
00521     {
00522 /// <summary>
00523 /// Describes the kind of plots that can be produced.
00524 /// </summary>
00525         public enum PlotType
00526         {
00527 /// <summary>
00528 /// A symbol is drawn at each sampled point, whose
00529 /// colour depends on the value of the function at that point.
00530 /// </summary>
00531             SampledPoints,
00532
00533 /// <summary>
00534 /// The plot area is tessellated with cells whose colour
00535 /// depends on the value of the function at a point within the cell.
00536 /// </summary>
00537             Tessellation,
00538
00539 /// <summary>
00540 /// A rasterised tessellation is created and then stretched and interpolated
00541 /// to fill the plot area.
00542 /// </summary>
00543             Raster
00544         }
00545
00546 /// <summary>
00547 /// The symbol to draw at the sampled points.
00548 /// </summary>
00549         public IDataPointElement SampledPointElement { get; set; } = new PathDataPointElement();
00550
00551 /// <summary>
00552 /// Resolution on the X axis for the rasterised tesselation.
00553 /// </summary>
00554         public int RasterResolutionX { get; set; } = 512;
00555
00556 /// <summary>
00557 /// Resolution on the Y axis for the rasterised tesselation.
00558 /// </summary>
00559         public int RasterResolutionY { get; set; } = 512;
00560
00561 /// <summary>
00562 /// The function to plot.
00563 /// </summary>
00564         public Function2DGrid Function { get; set; }
00565
00566 /// <summary>
00567 /// The kind of plot that is produced.
00568 /// </summary>
00569         public PlotType Type { get; set; } = PlotType.SampledPoints;
00570
00571 /// <summary>
00572 /// A function associating sampled function values to a <see cref="Colour"/>.  You should
00573 /// set this to a function accepting a single <see langword="double"/> argument ranging between
00574 /// 0 and 1, and returning the corresponding colour.  The default value returns black 0 and white for
      1.
00575 /// </summary>
00576         public Func<double, Colour> Colouring { get; set; } = x => { x = Math.Max(0, Math.Min(1, x));
      return Colour.FromRgb(x, x, x); };
00577
00578 /// <summary>
00579 /// A tag to identify the function in the plot.
00580 /// </summary>
00581         public string Tag { get; set; }
00582
00583 /// <summary>
00584 /// The coordinate system used to transform the points from data space to plot space.
00585 /// </summary>
00586         public IContinuousInvertibleCoordinateSystem CoordinateSystem { get; set; }
00587         ICoordinateSystem IPlotElement.CoordinateSystem => this.CoordinateSystem;
00588
00589 /// <summary>
00590 /// Create a new <see cref="Function2D"/> instance.
00591 /// </summary>
00592 /// <param name="function">The function to plot.</param>
00593 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00594         public Function2D(Function2DGrid function, IContinuousInvertibleCoordinateSystem
```

```
coordinateSystem)
00595         {
00596             this.Function = function;
00597             this.CoordinateSystem = coordinateSystem;
00598         }
00599
00600 /// <inheritdoc/>
00601         public unsafe void Plot(Graphics target)
00602         {
00603             if (Type == PlotType.SampledPoints)
00604             {
00605                 Point topLeft = CoordinateSystem.ToPlotCoordinates(new double[] { Function.MinX,
Function.MaxY });
00606                 Point bottomRight = CoordinateSystem.ToPlotCoordinates(new double[] { Function.MaxX,
Function.MinY });
00607
00608                 double size = Math.Min(Math.Abs(topLeft.X - bottomRight.X), Math.Abs(topLeft.Y -
bottomRight.Y)) * 0.0075;
00609
00610                 if (Function.Type != Function2DGrid.GridType.Irregular)
00611                 {
00612                     size = Math.Min(Math.Abs(topLeft.X - bottomRight.X) / Function.StepsX,
Math.Abs(topLeft.Y - bottomRight.Y) / Function.StepsY) * 0.4;
00613                 }
00614
00615                 for (int i = 0; i < Function.DataPoints.Count; i++)
00616                 {
00617                     Point pt = CoordinateSystem.ToPlotCoordinates(Function.DataPoints[i]);
00618                     Colour col = Colouring((Function.DataPoints[i][2] - Function.MinZ) /
(Function.MaxZ - Function.MinZ));
00619
00620                     string tag = Tag;
00621                     if (!string.IsNullOrEmpty(tag) && target.UseUniqueTags)
00622                     {
00623                         tag += "@" + i.ToString();
00624                     }
00625
00626                     target.Save();
00627                     target.Translate(pt);
00628                     target.Scale(size, size);
00629
00630                     SampledPointElement.Plot(target, new PlotElementPresentationAttributes() { Fill =
col, Stroke = null }, tag);
00631
00632                     target.Restore();
00633                 }
00634             }
00635             else if (Type == PlotType.Tessellation || (Type == PlotType.Raster && Function.Type ==
Function2DGrid.GridType.Irregular))
00636             {
00637                 double width = (Function.MaxX - Function.MinX) / Function.StepsX;
00638                 double height = (Function.MaxY - Function.MinY) / Function.StepsY;
00639
00640                 Point topLeft = CoordinateSystem.ToPlotCoordinates(new double[] { Function.MinX,
Function.MaxY });
00641                 Point topRight = CoordinateSystem.ToPlotCoordinates(new double[] { Function.MaxX,
Function.MaxY });
00642                 Point bottomRight = CoordinateSystem.ToPlotCoordinates(new double[] { Function.MaxX,
Function.MinY });
00643                 Point bottomLeft = CoordinateSystem.ToPlotCoordinates(new double[] { Function.MinX,
Function.MinY });
00644
00645
00646                 Graphics strokes = new Graphics();
00647                 Graphics fills = new Graphics();
00648
00649                 bool anyTransparent = false;
00650
00651                 if (Function.Type != Function2DGrid.GridType.Irregular)
00652                 {
00653                     for (int i = 0; i < Function.DataPoints.Count; i++)
00654                     {
00655                         Colour col = Colouring((Function.DataPoints[i][2] - Function.MinZ) /
(Function.MaxZ - Function.MinZ));
00656
00657                         if (col.A != 1)
00658                         {
00659                             anyTransparent = true;
00660                         }
00661
00662                         if (Function.Type == Function2DGrid.GridType.Rectangular)
00663                         {
00664                             Point p1 = CoordinateSystem.ToPlotCoordinates(new double[] {
Function.DataPoints[i][0] - width * 0.5, Function.DataPoints[i][1] - height * 0.5 });
00665                             Point p2 = CoordinateSystem.ToPlotCoordinates(new double[] {
Function.DataPoints[i][0] - width * 0.5, Function.DataPoints[i][1] + height * 0.5 });
00666                             Point p3 = CoordinateSystem.ToPlotCoordinates(new double[] {
```

```
Function.DataPoints[i][0] + width * 0.5, Function.DataPoints[i][1] + height * 0.5 });
00667                           Point p4 = CoordinateSystem.ToPlotCoordinates(new double[] {
Function.DataPoints[i][0] + width * 0.5, Function.DataPoints[i][1] - height * 0.5 });
00668
00669                           GraphicsPath pth = new
GraphicsPath().MoveTo(p1).LineTo(p2).LineTo(p3).LineTo(p4).Close();
00670
00671                           string tag = Tag;
00672                           string strokeTag = Tag;
00673                           if (!string.IsNullOrEmpty(tag) && target.UseUniqueTags)
00674                           {
00675                               tag += "@" + i.ToString();
00676                               strokeTag += "@stroke" + i.ToString();
00677                           }
00678
00679                           strokes.StrokePath(pth, col, 0.5, tag:  strokeTag);
00680                           fills.FillPath(pth, col, tag);
00681                       }
00682                       else if (Function.Type == Function2DGrid.GridType.HexagonHorizontal)
00683                       {
00684                           double rY = height * 0.57735026919;
00685                           double rX = width * 2 / 3;
00686
00687                           Point[] points = new Point[6];
00688
00689                           for (int j = 0; j < 6; j++)
00690                           {
00691                               points[j] = CoordinateSystem.ToPlotCoordinates(new double[] {
Function.DataPoints[i][0] + rX * Math.Cos(Math.PI / 3 * j), Function.DataPoints[i][1] + rY *
Math.Sin(Math.PI / 3 * j) });
00692                           }
00693
00694                           GraphicsPath pth = new
GraphicsPath().MoveTo(points[0]).LineTo(points[1]).LineTo(points[2]).LineTo(points[3]).LineTo(points[4]).LineTo(points[5
00695
00696                           string tag = Tag;
00697                           string strokeTag = Tag;
00698                           if (!string.IsNullOrEmpty(tag) && target.UseUniqueTags)
00699                           {
00700                               tag += "@" + i.ToString();
00701                               strokeTag += "@stroke" + i.ToString();
00702                           }
00703
00704                           strokes.StrokePath(pth, col, 0.5, tag:  strokeTag);
00705                           fills.FillPath(pth, col, tag);
00706                       }
00707                       else if (Function.Type == Function2DGrid.GridType.HexagonVertical)
00708                       {
00709                           double rX = width * 0.57735026919;
00710                           double rY = height * 2 / 3;
00711
00712                           Point[] points = new Point[6];
00713
00714                           for (int j = 0; j < 6; j++)
00715                           {
00716                               points[j] = CoordinateSystem.ToPlotCoordinates(new double[] {
Function.DataPoints[i][0] + rX * Math.Sin(Math.PI / 3 * j), Function.DataPoints[i][1] + rY *
Math.Cos(Math.PI / 3 * j) });
00717                           }
00718
00719                           GraphicsPath pth = new
GraphicsPath().MoveTo(points[0]).LineTo(points[1]).LineTo(points[2]).LineTo(points[3]).LineTo(points[4]).LineTo(points[5
00720
00721                           string tag = Tag;
00722                           string strokeTag = Tag;
00723                           if (!string.IsNullOrEmpty(tag) && target.UseUniqueTags)
00724                           {
00725                               tag += "@" + i.ToString();
00726                               strokeTag += "@stroke" + i.ToString();
00727                           }
00728
00729                           strokes.StrokePath(pth, col, 0.5, tag:  strokeTag);
00730                           fills.FillPath(pth, col, tag);
00731                       }
00732                   }
00733               }
00734               else
00735               {
00736                   double[][] plotCoordinates = new double[Function.DataPoints.Count][];
00737
00738                   double minX = double.MaxValue;
00739                   double minY = double.MaxValue;
00740                   double maxX = double.MinValue;
00741                   double maxY = double.MinValue;
00742
00743                   for (int i = 0; i < Function.DataPoints.Count; i++)
00744                   {
```

```
00745                         Point pt = CoordinateSystem.ToPlotCoordinates(Function.DataPoints[i]);
00746                         minX = Math.Min(minX, pt.X);
00747                         minY = Math.Min(minY, pt.Y);
00748                         maxX = Math.Max(maxX, pt.X);
00749                         maxY = Math.Max(maxY, pt.Y);
00750                         plotCoordinates[i] = new double[] { pt.X, pt.Y };
00751                     }
00752 
00753                     List<double[][]> cells = Voronoi.Voronoi.GetVoronoiCells(plotCoordinates, minX,
      minY, maxX, maxY);
00754 
00755                     for (int j = 0; j < cells.Count; j++)
00756                     {
00757                         Colour col = Colouring((Function.DataPoints[j][2] - Function.MinZ) /
      (Function.MaxZ - Function.MinZ));
00758 
00759                         if (col.A != 1)
00760                         {
00761                             anyTransparent = true;
00762                         }
00763 
00764                         GraphicsPath path = new GraphicsPath();
00765 
00766                         for (int k = 0; k < cells[j].Length; k++)
00767                         {
00768                             path.LineTo(cells[j][k][0], cells[j][k][1]);
00769                         }
00770 
00771                         path.Close();
00772 
00773                         string tag = Tag;
00774                         string strokeTag = Tag;
00775                         if (!string.IsNullOrEmpty(tag) && target.UseUniqueTags)
00776                         {
00777                             tag += "@" + j.ToString();
00778                             strokeTag += "@stroke" + j.ToString();
00779                         }
00780 
00781                         strokes.StrokePath(path, col, 0.5, tag:  strokeTag);
00782                         fills.FillPath(path, col, tag);
00783 
00784                     }
00785                 }
00786 
00787                 target.Save();
00788                 target.SetClippingPath(new
      GraphicsPath().MoveTo(topLeft).LineTo(topRight).LineTo(bottomRight).LineTo(bottomLeft).Close());
00789 
00790                 if (Type != PlotType.Raster)
00791                 {
00792                     if (!anyTransparent)
00793                     {
00794                         target.DrawGraphics(0, 0, strokes);
00795                     }
00796 
00797                     target.DrawGraphics(0, 0, fills);
00798                 }
00799                 else
00800                 {
00801                     double x1 = Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X,
      bottomRight.X));
00802                     double x0 = Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X,
      bottomRight.X));
00803 
00804                     double y1 = Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y,
      bottomRight.Y));
00805                     double y0 = Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y,
      bottomRight.Y));
00806 
00807                     double scaleX = RasterResolutionX / (x1 - x0);
00808                     double scaleY = RasterResolutionY / (y1 - y0);
00809 
00810                     Graphics toBeRasterised = new Graphics();
00811                     toBeRasterised.Scale(scaleX, scaleY);
00812 
00813                     if (!anyTransparent)
00814                     {
00815                         toBeRasterised.DrawGraphics(0, 0, strokes);
00816                     }
00817 
00818                     toBeRasterised.DrawGraphics(0, 0, fills);
00819 
00820                     if (toBeRasterised.TryRasterise(new Rectangle(x0 * scaleX, y0 * scaleY, (x1 - x0)
      * scaleX, (y1 - y0) * scaleY), 1, true, out RasterImage image))
00821                     {
00822                         target.DrawRasterImage(x0, y0, x1 - x0, y1 - y0, image, Tag);
00823                     }
```

```
00824                 }
00825                 target.Restore();
00826
00827             }
00828             else if (Type == PlotType.Raster)
00829             {
00830                 Point topLeft = CoordinateSystem.ToPlotCoordinates(new double[] { Function.MinX,
      Function.MaxY });
00831                 Point topRight = CoordinateSystem.ToPlotCoordinates(new double[] { Function.MaxX,
      Function.MaxY });
00832                 Point bottomRight = CoordinateSystem.ToPlotCoordinates(new double[] { Function.MaxX,
      Function.MinY });
00833                 Point bottomLeft = CoordinateSystem.ToPlotCoordinates(new double[] { Function.MinX,
      Function.MinY });
00834
00835                 double width = Math.Abs(topLeft.X - bottomRight.X);
00836                 double height = Math.Abs(topLeft.Y - bottomRight.Y);
00837
00838                 RasterImage image = null;
00839
00840                 Function2DGrid function = Function;
00841
00842                 if (Function.Type != Function2DGrid.GridType.Rectangular)
00843                 {
00844                     function = Function.ToRectangular();
00845                 }
00846
00847
00848                 double[,] imageData = new double[function.StepsX + 1, function.StepsY + 1];
00849
00850                 int sampleWidth = function.StepsX + 1;
00851                 int sampleHeight = function.StepsY + 1;
00852
00853                 for (int i = 0; i < function.DataPoints.Count; i++)
00854                 {
00855                     int x = (int)Math.Round((function.DataPoints[i][0] - function.MinX) /
      (function.MaxX - function.MinX) * function.StepsX);
00856                     int y = (int)Math.Round((function.DataPoints[i][1] - function.MinY) /
      (function.MaxY - function.MinY) * function.StepsY);
00857
00858                     imageData[x, y] = (function.DataPoints[i][2] - function.MinZ) / (function.MaxZ -
      function.MinZ);
00859                 }
00860
00861                 IntPtr imageAddr = Marshal.AllocHGlobal(RasterResolutionX * RasterResolutionY * 4);
00862                 DisposableIntPtr disp = new DisposableIntPtr(imageAddr);
00863
00864                 unsafe
00865                 {
00866                     byte* imageBytes = (byte*)imageAddr;
00867
00868                     for (int y = 0; y < RasterResolutionY; y++)
00869                     {
00870                         for (int x = 0; x < RasterResolutionX; x++)
00871                         {
00872                             double realX = (double)x / (RasterResolutionX - 1);
00873                             double realY = 1 - (double)y / (RasterResolutionY - 1);
00874
00875                             double intensity00 = imageData[(int)Math.Floor(realX * (sampleWidth - 1)),
      (int)Math.Floor(realY * (sampleHeight - 1))];
00876                             double intensity01 = imageData[(int)Math.Floor(realX * (sampleWidth - 1)),
      Math.Min(sampleHeight - 1, (int)Math.Floor(realY * (sampleHeight - 1)) + 1)];
00877                             double intensity10 = imageData[Math.Min(sampleWidth - 1,
      (int)Math.Floor(realX * (sampleWidth - 1)) + 1), (int)Math.Floor(realY * (sampleHeight - 1))];
00878                             double intensity11 = imageData[Math.Min(sampleWidth - 1,
      (int)Math.Floor(realX * (sampleWidth - 1)) + 1), Math.Min(sampleHeight - 1, (int)Math.Floor(realY *
      (sampleHeight - 1)) + 1)];
00879
00880                             double fracX = realX * (sampleWidth - 1) - Math.Floor(realX * (sampleWidth
      - 1));
00881                             double fracY = realY * (sampleHeight - 1) - Math.Floor(realY *
      (sampleHeight - 1));
00882
00883                             double intensityX0 = intensity00 * (1 - fracX) + intensity10 * fracX;
00884                             double intensityX1 = intensity01 * (1 - fracX) + intensity11 * fracX;
00885                             double intensity = intensityX0 * (1 - fracY) + intensityX1 * fracY;
00886
00887                             Colour col = Colouring(intensity);
00888
00889                             imageBytes[(y * RasterResolutionX + x) * 4] = (byte)(col.R * 255);
00890                             imageBytes[(y * RasterResolutionX + x) * 4 + 1] = (byte)(col.G * 255);
00891                             imageBytes[(y * RasterResolutionX + x) * 4 + 2] = (byte)(col.B * 255);
00892                             imageBytes[(y * RasterResolutionX + x) * 4 + 3] = (byte)(col.A * 255);
00893                         }
00894                     }
00895                 }
00896
```

```
00897                 image = new RasterImage(ref disp, RasterResolutionX, RasterResolutionY, true, true);
00898
00899
00900                 if (image != null)
00901                 {
00902                     target.Save();
00903                     target.SetClippingPath(new
      GraphicsPath().MoveTo(topLeft).LineTo(topRight).LineTo(bottomRight).LineTo(bottomLeft).Close());
00904
00905                     target.DrawRasterImage(topLeft, new Size(width, height), image, Tag);
00906
00907                     target.Restore();
00908                 }
00909             }
00910         }
00911     }
00912 }
00913
```

8.42 Pie.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021
00022 namespace VectSharp.Plots
00023 {
00024 /// <summary>
00025 /// A plot element that draws a pie or a doughnut.
00026 /// </summary>
00027     public class Pie :  IPlotElement
00028     {
00029 /// <summary>
00030 /// The data in the pie chart.  The values do not need to be normalised.
00031 /// </summary>
00032         public IReadOnlyList<double> Data { get; set; }
00033
00034 /// <summary>
00035 /// The centre of the pie/doughnut, in data coordinates.
00036 /// </summary>
00037         public IReadOnlyList<double> Centre { get; set; }
00038
00039 /// <summary>
00040 /// The outer radius of the pie/doughnut, in data coordinates.
00041 /// </summary>
00042         public IReadOnlyList<double> OuterRadius { get; set; }
00043
00044 /// <summary>
00045 /// The inner radius of the doughnut, in data coordinates.  Set to [0, 0] for a pie chart.
00046 /// </summary>
00047         public IReadOnlyList<double> InnerRadius { get; set; }
00048
00049 /// <summary>
00050 /// The initial angle starting from which the pie slices are drawn.
00051 /// </summary>
00052         public double StartAngle { get; set; } = 0;
00053
00054 /// <summary>
00055 /// Determines whether the slices are drawn in clockwise or anti-clockwise fashion.
00056 /// </summary>
00057         public bool Clockwise { get; set; } = false;
00058
00059 /// <summary>
00060 /// The coordinate system used to transform the points from data space to plot space.
00061 /// </summary>
00062         public ICoordinateSystem<IReadOnlyList<double>> CoordinateSystem { get; set; }
```

```
00063         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00064
00065 /// <summary>
00066 /// Presentation attributes for the slices.  An element from this collection is used for each slice in
00067 /// the pie/doughnut; if there are more slices than elements in this collection, the presentation
      attributes
00068 /// are wrapped.
00069 /// </summary>
00070         public IReadOnlyList<PlotElementPresentationAttributes> PresentationAttributes { get; set; } =
      new PlotElementPresentationAttributes[] { new PlotElementPresentationAttributes() };
00071
00072 /// <summary>
00073 /// A tag to identify the pie/doughnut in the plot.
00074 /// </summary>
00075         public string Tag { get; set; }
00076
00077 /// <summary>
00078 /// Create a new <see cref="Pie"/> instance drawing a pie chart.
00079 /// </summary>
00080 /// <param name="data">The data in the pie chart.  The values do not need to be normalised.</param>
00081 /// <param name="centre">The centre of the pie, in data coordinates.</param>
00082 /// <param name="radius">The radius of the pie, in data coordinates.</param>
00083 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00084         public Pie(IReadOnlyList<double> data, IReadOnlyList<double> centre, IReadOnlyList<double>
      radius, ICoordinateSystem<IReadOnlyList<double» coordinateSystem)
00085         {
00086             Data = data;
00087             Centre = centre;
00088             OuterRadius = radius;
00089             InnerRadius = new double[] { 0, 0 };
00090             CoordinateSystem = coordinateSystem;
00091         }
00092
00093 /// <summary>
00094 /// Create a new <see cref="Pie"/> instance drawing a doughnut chart.
00095 /// </summary>
00096 /// <param name="data">The data in the doughnut chart.  The values do not need to be
      normalised.</param>
00097 /// <param name="centre">The centre of the doughnut, in data coordinates.</param>
00098 /// <param name="innerRadius">The inner radius of the doughnut, in data coordinates.</param>
00099 /// <param name="outerRadius">The outer radius of the doughnut, in data coordinates.</param>
00100 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00101         public Pie(IReadOnlyList<double> data, IReadOnlyList<double> centre, IReadOnlyList<double>
      innerRadius, IReadOnlyList<double> outerRadius, ICoordinateSystem<IReadOnlyList<double»
      coordinateSystem)
00102         {
00103             Data = data;
00104             Centre = centre;
00105             OuterRadius = outerRadius;
00106             InnerRadius = innerRadius;
00107             CoordinateSystem = coordinateSystem;
00108         }
00109
00110 /// <inheritdoc/>
00111         public void Plot(Graphics target)
00112         {
00113             double total = Data.Sum();
00114
00115             double currAngle = StartAngle;
00116
00117             double minAngle = Data.Where(x => x > 0).Min() / total * 2 * Math.PI;
00118
00119             minAngle = Math.Min(minAngle, 1);
00120             minAngle = Math.Max(minAngle, 0.1);
00121
00122             for (int i = 0; i < Data.Count; i++)
00123             {
00124                 double endAngle = currAngle + Data[i] / total * 2 * Math.PI;
00125
00126                 GraphicsPath pth = new GraphicsPath();
00127
00128                 if (InnerRadius[0] != 0 || InnerRadius[1] != 0)
00129                 {
00130                     Point[] innerCircle = new Point[(int)((endAngle - currAngle) / minAngle * 10) +
      1];
00131
00132                     Point[] outerCircle = new Point[(int)((endAngle - currAngle) / minAngle * 10) +
      1];
00133
00134                     double step = (endAngle - currAngle) / (innerCircle.Length - 1);
00135
00136                     for (int j = 0; j < innerCircle.Length; j++)
00137                     {
00138                         double currAng = currAngle + step * j;
00139
```

```
00140                         if (Clockwise)
00141                         {
00142                             currAng *= -1;
00143                         }
00144 
00145                         innerCircle[j] = CoordinateSystem.ToPlotCoordinates(new double[] { Centre[0] +
      InnerRadius[0] * Math.Cos(currAng), Centre[1] + InnerRadius[1] * Math.Sin(currAng) });
00146                         outerCircle[outerCircle.Length - 1 - j] =
      CoordinateSystem.ToPlotCoordinates(new double[] { Centre[0] + OuterRadius[0] * Math.Cos(currAng),
      Centre[1] + OuterRadius[1] * Math.Sin(currAng) });
00147                     }
00148 
00149                     currAngle = endAngle;
00150 
00151                     pth.AddSmoothSpline(innerCircle);
00152 
00153                     pth.LineTo(outerCircle[0]);
00154                     pth.AddSmoothSpline(outerCircle);
00155 
00156                     pth.Close();
00157                 }
00158                 else
00159                 {
00160                     Point[] outerCircle = new Point[(int)((endAngle - currAngle) / minAngle * 10) +
      1];
00161 
00162                     double step = (endAngle - currAngle) / (outerCircle.Length - 1);
00163 
00164                     for (int j = 0; j < outerCircle.Length; j++)
00165                     {
00166                         double currAng = currAngle + step * j;
00167 
00168                         if (Clockwise)
00169                         {
00170                             currAng *= -1;
00171                         }
00172 
00173                         outerCircle[outerCircle.Length - 1 - j] =
      CoordinateSystem.ToPlotCoordinates(new double[] { Centre[0] + OuterRadius[0] * Math.Cos(currAng),
      Centre[1] + OuterRadius[1] * Math.Sin(currAng) });
00174                     }
00175 
00176                     currAngle = endAngle;
00177 
00178                     pth.MoveTo(CoordinateSystem.ToPlotCoordinates(Centre));
00179 
00180                     pth.LineTo(outerCircle[0]);
00181                     pth.AddSmoothSpline(outerCircle);
00182 
00183                     pth.Close();
00184                 }
00185 
00186                 PlotElementPresentationAttributes attributes = PresentationAttributes[i %
      PresentationAttributes.Count];
00187 
00188                 string tag = Tag;
00189                 string strokeTag = tag;
00190 
00191                 if (target.UseUniqueTags && !string.IsNullOrEmpty(tag))
00192                 {
00193                     tag += "@" + i.ToString();
00194                     strokeTag += "@" + i.ToString() + "_stroke";
00195                 }
00196 
00197                 if (attributes.Fill != null)
00198                 {
00199                     target.FillPath(pth, attributes.Fill, tag);
00200                 }
00201 
00202                 if (attributes.Stroke != null)
00203                 {
00204                     target.StrokePath(pth, attributes.Stroke, attributes.LineWidth,
      attributes.LineCap, attributes.LineJoin, attributes.LineDash, strokeTag);
00205                 }
00206             }
00207         }
00208     }
00209 }
```

8.43 Plot.Area.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
```

```
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021
00022 namespace VectSharp.Plots
00023 {
00024     partial class Plot
00025     {
00026         partial class Create
00027         {
00028 /// <summary>
00029 /// Create a new area chart.
00030 /// </summary>
00031 /// <param name="data">The data to plot.</param>
00032 /// <param name="vertical">If this is <see langword="true"/> (the default), the highlighted area goes from the X axis up to the sampled values.  If this is <see langword="false"/>, the highlighted area goes from the Y axis to the sampled values.</param>
00033 /// <param name="smooth">If this is <see langword="false"/> (the default), the sampled values are joined by a polyline.  If this is <see langword="true"/>, they are joined by a smooth spline passing through them.</param>
00034 /// <param name="width">The width of the plot.</param>
00035 /// <param name="height">The height of the plot.</param>
00036 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00037 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00038 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00039 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00040 /// <param name="xAxisTitle">Title for the X axis.</param>
00041 /// <param name="yAxisTitle">Title for the Y axis.</param>
00042 /// <param name="title">Title for the plot.</param>
00043 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00044 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00045 /// <param name="dataPresentationAttributes">Presentation attributes for the plotted data.</param>
00046 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space to plot space.</param>
00047 /// <returns>A <see cref="Plot"/> containing the area chart.</returns>
00048             public static Plot AreaChart(IReadOnlyList<IReadOnlyList<double>> data, bool vertical = true, bool smooth = false, double width = 350, double height = 250,
00049                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00050                 double axisArrowSize = 3,
00051                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00052                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00053                 string xAxisTitle = null,
00054                 string yAxisTitle = null,
00055                 string title = null,
00056                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00057                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00058                 PlotElementPresentationAttributes dataPresentationAttributes = null,
00059                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00060             {
00061                 if (axisPresentationAttributes == null)
00062                 {
00063                     axisPresentationAttributes = new PlotElementPresentationAttributes();
00064                 }
00065
00066                 if (gridPresentationAttributes == null)
00067                 {
00068                     gridPresentationAttributes = new PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(220, 220, 220)) };
00069                 }
00070
00071                 if (axisLabelPresentationAttributes == null)
00072                 {
00073                     axisLabelPresentationAttributes = new PlotElementPresentationAttributes() { Stroke = null };
00074                 }
00075
00076                 if (axisTitlePresentationAttributes == null)
00077                 {
00078                     axisTitlePresentationAttributes = new PlotElementPresentationAttributes() { Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 14), Stroke = null };
00079                 }
```

```
00080
00081                 if (dataPresentationAttributes == null)
00082                 {
00083                     dataPresentationAttributes = new PlotElementPresentationAttributes()
00084                     {
00085                         Fill = new SolidColourBrush(Colour.FromRgb(197, 235, 255)),
00086                         Stroke = new SolidColourBrush(Colour.FromRgb(0, 114, 178))
00087                     };
00088                 }
00089
00090                 if (titlePresentationAttributes == null)
00091                 {
00092                     titlePresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 18)
      };
00093                 }
00094
00095                 double baselineValue = 0;
00096
00097                 if (coordinateSystem is LogarithmicCoordinateSystem2D || (!vertical &&
      coordinateSystem is LinLogCoordinateSystem2D) || (vertical && coordinateSystem is
      LogLinCoordinateSystem2D))
00098                 {
00099                     baselineValue = 1;
00100                 }
00101
00102                 Func<IReadOnlyList<double>, IReadOnlyList<double» getBaseline;
00103
00104                 if (vertical)
00105                 {
00106                     getBaseline = x => new double[] { x[0], baselineValue };
00107                 }
00108                 else
00109                 {
00110                     getBaseline = x => new double[] { baselineValue, x[1] };
00111                 }
00112
00113                 IReadOnlyList<double>[] allData = new IReadOnlyList<double>[data.Count * 2];
00114
00115                 for (int i = 0; i < data.Count; i++)
00116                 {
00117                     allData[i] = data[i];
00118                     allData[i + data.Count] = getBaseline(data[i]);
00119                 }
00120
00121                 (double minX, double minY, double maxX, double maxY, double rangeX, double rangeY) =
      GetDataRange(allData);
00122
00123                 if (coordinateSystem == null)
00124                 {
00125                     coordinateSystem = new LinearCoordinateSystem2D(allData, width, height);
00126                 }
00127
00128                 Point topLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, maxY });
00129                 Point topRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, maxY });
00130                 Point bottomRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, minY });
00131                 Point bottomLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, minY });
00132
00133                 double[] marginTopLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00134                 double[] marginTopRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00135                 double[] marginBottomRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00136                 double[] marginBottomLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00137
00138                 double[] p1 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00139                 double[] p2 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00140                 double[] p3 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00141                 double[] p4 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00142
00143
00144                 double[] p5 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
```

```
    Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00145                 double[] p6 = coordinateSystem.ToDataCoordinates(new
    Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
    Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00146                 double[] p7 = coordinateSystem.ToDataCoordinates(new
    Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
    Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00147                 double[] p8 = coordinateSystem.ToDataCoordinates(new
    Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
    Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00148
00149                 Grid xGrid = new Grid(p1, p2, p3, p4, coordinateSystem) { IntervalCount = 5,
    PresentationAttributes = gridPresentationAttributes };
00150                 Grid yGrid = new Grid(p5, p6, p7, p8, coordinateSystem) { IntervalCount = 5,
    PresentationAttributes = gridPresentationAttributes };
00151
00152                 ContinuousAxis xAxis = new ContinuousAxis(marginBottomLeft[0] < marginBottomRight[0] ?
    marginBottomLeft :  marginBottomRight, marginBottomLeft[0] < marginBottomRight[0] ?  marginBottomRight
    :  coordinateSystem.ToDataCoordinates(coordinateSystem.ToPlotCoordinates(marginBottomLeft) + new
    Point(-axisArrowSize - 7, 0)), coordinateSystem) { PresentationAttributes =
    axisPresentationAttributes, ArrowSize = axisArrowSize };
00153                 ContinuousAxis yAxis = new ContinuousAxis(marginBottomLeft[1] < marginTopLeft[1] ?
    marginBottomLeft :  marginTopLeft, marginBottomLeft[1] < marginTopLeft[1] ?  marginTopLeft :
    coordinateSystem.ToDataCoordinates(coordinateSystem.ToPlotCoordinates(marginBottomLeft) + new Point(0,
    axisArrowSize + 7)), coordinateSystem) { PresentationAttributes = axisPresentationAttributes,
    ArrowSize = axisArrowSize };
00154
00155                 ContinuousAxisTicks xTicks = new ContinuousAxisTicks(p3, p4, coordinateSystem) {
    PresentationAttributes = axisPresentationAttributes };
00156                 ContinuousAxisTicks yTicks = new ContinuousAxisTicks(p6, p5, coordinateSystem) {
    PresentationAttributes = axisPresentationAttributes };
00157
00158                 ContinuousAxisLabels xLabels = new ContinuousAxisLabels(p3, p4, coordinateSystem) {
    PresentationAttributes = axisLabelPresentationAttributes, Alignment = TextAnchors.Center, Baseline =
    TextBaselines.Top, Rotation = 0, IntervalCount = 5 };
00159                 ContinuousAxisLabels yLabels = new ContinuousAxisLabels(p6, p5, coordinateSystem) {
    PresentationAttributes = axisLabelPresentationAttributes, Position = _ => -10, Alignment =
    TextAnchors.Right, Rotation = 0, IntervalCount = 5 };
00160
00161                 Graphics xLabelsSize = new Graphics();
00162                 xLabels.Plot(xLabelsSize);
00163                 double xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
00164
00165                 Graphics yLabelsSize = new Graphics();
00166                 yLabels.Plot(yLabelsSize);
00167                 double yLabelsWidth = yLabelsSize.GetBounds().Size.Width;
00168
00169                 ContinuousAxisTitle xTitle = new ContinuousAxisTitle(xAxisTitle, marginBottomLeft,
    marginBottomRight, coordinateSystem, axisTitlePresentationAttributes) { Position = xLabelsHeight + 20,
    Alignment = TextAnchors.Center };
00170                 ContinuousAxisTitle yTitle = new ContinuousAxisTitle(yAxisTitle, marginBottomLeft,
    marginTopLeft, coordinateSystem, axisTitlePresentationAttributes) { Position = -20 - yLabelsWidth,
    Baseline = TextBaselines.Bottom, Alignment = TextAnchors.Center };
00171
00172                 Area<IReadOnlyList<double>> area = new Area<IReadOnlyList<double>>(data, getBaseline,
    coordinateSystem) { PresentationAttributes = dataPresentationAttributes, Smooth = smooth };
00173
00174                 TextLabel<IReadOnlyList<double>> titleLabel = new
    TextLabel<IReadOnlyList<double>>(title,
    coordinateSystem.ToDataCoordinates((coordinateSystem.ToPlotCoordinates(marginTopLeft) +
    coordinateSystem.ToPlotCoordinates(marginTopRight)) * 0.5 + new Point(0, -10)), coordinateSystem) {
    Baseline = TextBaselines.Bottom, PresentationAttributes = titlePresentationAttributes };
00175
00176                 Plot tbr = new Plot();
00177                 tbr.AddPlotElements(xGrid, yGrid, xAxis, yAxis, xTicks, yTicks, xLabels, yLabels,
    xTitle, yTitle, area, titleLabel);
00178
00179                 return tbr;
00180             }
00181
00182
00183 /// <summary>
00184 /// Create a new stacked area chart.
00185 /// </summary>
00186 /// <param name="data">The data to plot.</param>
00187 /// <param name="vertical">If this is <see langword="true"/> (the default), the highlighted area goes
    from the X axis up to the sampled values.  If this is <see langword="false"/>, the highlighted area
    goes from the Y axis to the sampled values.</param>
00188 /// <param name="smooth">If this is <see langword="false"/> (the default), the sampled values are
    joined by a polyline.  If this is <see langword="true"/>, they are joined by a smooth spline passing
    through them.</param>
00189 /// <param name="width">The width of the plot.</param>
00190 /// <param name="height">The height of the plot.</param>
00191 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00192 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00193 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00194 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
```

```
00195 /// <param name="xAxisTitle">Title for the X axis.</param>
00196 /// <param name="yAxisTitle">Title for the Y axis.</param>
00197 /// <param name="title">Title for the plot.</param>
00198 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00199 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00200 /// <param name="dataPresentationAttributes">Presentation attributes for the plotted data.</param>
00201 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00202 /// <returns>A <see cref="Plot"/> containing the stacked area chart.</returns>
00203             public static Plot StackedAreaChart(IReadOnlyList<IReadOnlyList<double» data, bool
      vertical = true, bool smooth = false, double width = 350, double height = 250,
00204                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00205                 double axisArrowSize = 3,
00206                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00207                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00208                 string xAxisTitle = null,
00209                 string yAxisTitle = null,
00210                 string title = null,
00211                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00212                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00213                 IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
00214                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00215             {
00216                 if (axisPresentationAttributes == null)
00217                 {
00218                     axisPresentationAttributes = new PlotElementPresentationAttributes();
00219                 }
00220 
00221                 if (gridPresentationAttributes == null)
00222                 {
00223                     gridPresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      new SolidColourBrush(Colour.FromRgb(220, 220, 220)) };
00224                 }
00225 
00226                 if (axisLabelPresentationAttributes == null)
00227                 {
00228                     axisLabelPresentationAttributes = new PlotElementPresentationAttributes() { Stroke
      = null };
00229                 }
00230 
00231                 if (axisTitlePresentationAttributes == null)
00232                 {
00233                     axisTitlePresentationAttributes = new PlotElementPresentationAttributes() { Font =
      new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 14), Stroke =
      null };
00234                 }
00235 
00236                 if (dataPresentationAttributes == null)
00237                 {
00238                     dataPresentationAttributes = new PlotElementPresentationAttributes[] { new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(197, 235, 255)),
      Stroke = new SolidColourBrush(Colour.FromRgb(0, 114, 178)) },
00239                                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 233, 218)),
      Stroke = new SolidColourBrush(Colour.FromRgb(213, 94, 0)) },
00240                                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 222, 240)),
      Stroke = new SolidColourBrush(Colour.FromRgb(204, 121, 167)) },
00241                                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 242, 216)),
      Stroke = new SolidColourBrush(Colour.FromRgb(230, 159, 0)) },
00242                                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(214, 241, 255)),
      Stroke = new SolidColourBrush(Colour.FromRgb(86, 180, 233)) },
00243                                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(203, 255, 239)),
      Stroke = new SolidColourBrush(Colour.FromRgb(0, 158, 115)) },
00244                                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 249, 189)),
      Stroke = new SolidColourBrush(Colour.FromRgb(240, 228, 66)) } };
00245                 }
00246 
00247                 if (titlePresentationAttributes == null)
00248                 {
00249                     titlePresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 18)
      };
00250                 }
00251 
00252                 double baselineValue = 0;
00253 
00254                 if (coordinateSystem is LogarithmicCoordinateSystem2D || (!vertical &&
      coordinateSystem is LinLogCoordinateSystem2D) || (vertical && coordinateSystem is
      LogLinCoordinateSystem2D))
00255                 {
00256                     baselineValue = 1;
00257                 }
```

```
00258
00259                 Func<IReadOnlyList<double>, IReadOnlyList<double» getBaseline;
00260
00261                 if (vertical)
00262                 {
00263                     getBaseline = x => new double[] { x[0], baselineValue };
00264                 }
00265                 else
00266                 {
00267                     getBaseline = x => new double[] { baselineValue, x[1] };
00268                 }
00269
00270                 List<IReadOnlyList<double» allData = new List<IReadOnlyList<double»(data.Count *
      data[0].Count);
00271
00272                 allData.AddRange(data);
00273
00274                 List<List<IReadOnlyList<double»> groupedData = new
      List<List<IReadOnlyList<double»>(data[0].Count - 1);
00275
00276                 List<Dictionary<double, double» baselines = new List<Dictionary<double,
      double»(data[0].Count - 2);
00277
00278                 for (int i = 0; i < data.Count; i++)
00279                 {
00280                     allData.Add(getBaseline(data[i]));
00281                 }
00282
00283                 for (int i = 0; i < data[0].Count - 1; i++)
00284                 {
00285                     if (vertical)
00286                     {
00287                         groupedData.Add(new List<IReadOnlyList<double»(from el in data select new
      double[] { el[0], el[i + 1] }));
00288
00289                         if (i > 0)
00290                         {
00291                             Dictionary<double, double> baseline = new Dictionary<double,
      double>(data.Count);
00292
00293                             for (int j = 0; j < data.Count; j++)
00294                             {
00295                                 baseline[data[j][0]] = data[j][i];
00296                             }
00297
00298                             baselines.Add(baseline);
00299                         }
00300                     }
00301                     else
00302                     {
00303                         if (i == 0)
00304                         {
00305                             groupedData.Add(new List<IReadOnlyList<double»(from el in data select new
      double[] { el[0], el[1] }));
00306                         }
00307                         else
00308                         {
00309                             groupedData.Add(new List<IReadOnlyList<double»(from el in data select new
      double[] { el[i + 1], el[1] }));
00310
00311                             Dictionary<double, double> baseline = new Dictionary<double,
      double>(data.Count);
00312
00313                             if (i == 1)
00314                             {
00315                                 for (int j = 0; j < data.Count; j++)
00316                                 {
00317                                     baseline[data[j][1]] = data[j][0];
00318                                 }
00319                             }
00320                             else
00321                             {
00322                                 for (int j = 0; j < data.Count; j++)
00323                                 {
00324                                     baseline[data[j][1]] = data[j][i];
00325                                 }
00326                             }
00327
00328                             baselines.Add(baseline);
00329                         }
00330                     }
00331
00332                     allData.AddRange(groupedData[groupedData.Count - 1]);
00333                 }
00334
00335                 (double minX, double minY, double maxX, double maxY, double rangeX, double rangeY) =
      GetDataRange(allData);
```

```
00336
00337                 if (coordinateSystem == null)
00338                 {
00339                     coordinateSystem = new LinearCoordinateSystem2D(allData, width, height);
00340                 }
00341
00342                 Point topLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, maxY });
00343                 Point topRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, maxY });
00344                 Point bottomRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, minY });
00345                 Point bottomLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, minY });
00346
00347                 double[] marginTopLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00348                 double[] marginTopRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00349                 double[] marginBottomRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00350                 double[] marginBottomLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00351
00352                 double[] p1 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00353                 double[] p2 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00354                 double[] p3 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00355                 double[] p4 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00356
00357
00358                 double[] p5 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00359                 double[] p6 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00360                 double[] p7 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00361                 double[] p8 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00362
00363                 Grid xGrid = new Grid(p1, p2, p3, p4, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00364                 Grid yGrid = new Grid(p5, p6, p7, p8, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00365
00366                 ContinuousAxis xAxis = new ContinuousAxis(marginBottomLeft[0] < marginBottomRight[0] ?
      marginBottomLeft :  marginBottomRight, marginBottomLeft[0] < marginBottomRight[0] ?  marginBottomRight
      :  coordinateSystem.ToDataCoordinates(coordinateSystem.ToPlotCoordinates(marginBottomLeft) + new
      Point(-axisArrowSize - 7, 0)), coordinateSystem) { PresentationAttributes =
      axisPresentationAttributes, ArrowSize = axisArrowSize };
00367                 ContinuousAxis yAxis = new ContinuousAxis(marginBottomLeft[1] < marginTopLeft[1] ?
      marginBottomLeft :  marginTopLeft, marginBottomLeft[1] < marginTopLeft[1] ?  marginTopLeft :
      coordinateSystem.ToDataCoordinates(coordinateSystem.ToPlotCoordinates(marginBottomLeft) + new Point(0,
      axisArrowSize + 7)), coordinateSystem) { PresentationAttributes = axisPresentationAttributes,
      ArrowSize = axisArrowSize };
00368
00369                 ContinuousAxisTicks xTicks = new ContinuousAxisTicks(p3, p4, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00370                 ContinuousAxisTicks yTicks = new ContinuousAxisTicks(p6, p5, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00371
00372                 ContinuousAxisLabels xLabels = new ContinuousAxisLabels(p3, p4, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Alignment = TextAnchors.Center, Baseline =
      TextBaselines.Top, Rotation = 0, IntervalCount = 5 };
00373                 ContinuousAxisLabels yLabels = new ContinuousAxisLabels(p6, p5, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Position = _ => -10, Alignment =
      TextAnchors.Right, Rotation = 0, IntervalCount = 5 };
00374
00375                 Graphics xLabelsSize = new Graphics();
00376                 xLabels.Plot(xLabelsSize);
00377                 double xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
00378
00379                 Graphics yLabelsSize = new Graphics();
00380                 yLabels.Plot(yLabelsSize);
00381                 double yLabelsWidth = yLabelsSize.GetBounds().Size.Width;
00382
```

```
00383                 ContinuousAxisTitle xTitle = new ContinuousAxisTitle(xAxisTitle, marginBottomLeft,
      marginBottomRight, coordinateSystem, axisTitlePresentationAttributes) { Position = xLabelsHeight + 20,
      Alignment = TextAnchors.Center };
00384                 ContinuousAxisTitle yTitle = new ContinuousAxisTitle(yAxisTitle, marginBottomLeft,
      marginTopLeft, coordinateSystem, axisTitlePresentationAttributes) { Position = -20 - yLabelsWidth,
      Baseline = TextBaselines.Bottom, Alignment = TextAnchors.Center };
00385
00386
00387                 List<IPlotElement> areas = new List<IPlotElement>(groupedData.Count);
00388                 List<IPlotElement> lines = new List<IPlotElement>(groupedData.Count);
00389
00390                 for (int i = 0; i < groupedData.Count; i++)
00391                 {
00392                     if (i == 0)
00393                     {
00394                         Area<IReadOnlyList<double>> area = new
      Area<IReadOnlyList<double>>(groupedData[i], getBaseline, coordinateSystem) { PresentationAttributes =
      new PlotElementPresentationAttributes(dataPresentationAttributes[i %
      dataPresentationAttributes.Count]) { Stroke = null }, Smooth = smooth };
00395                         DataLine<IReadOnlyList<double>> line = new
      DataLine<IReadOnlyList<double>>(groupedData[i], coordinateSystem) { PresentationAttributes = new
      PlotElementPresentationAttributes(dataPresentationAttributes[i % dataPresentationAttributes.Count]) {
      Fill = null }, Smooth = smooth };
00396                         areas.Add(area);
00397                         lines.Add(line);
00398                     }
00399                     else
00400                     {
00401                         int index = i;
00402
00403                         if (vertical)
00404                         {
00405                             Area<IReadOnlyList<double>> area = new
      Area<IReadOnlyList<double>>(groupedData[i], x => new double[] { x[0], baselines[index - 1][x[0]] },
      coordinateSystem) { PresentationAttributes = new
      PlotElementPresentationAttributes(dataPresentationAttributes[i % dataPresentationAttributes.Count]) {
      Stroke = null }, Smooth = smooth };
00406                             DataLine<IReadOnlyList<double>> line = new
      DataLine<IReadOnlyList<double>>(groupedData[i], coordinateSystem) { PresentationAttributes = new
      PlotElementPresentationAttributes(dataPresentationAttributes[i % dataPresentationAttributes.Count]) {
      Fill = null }, Smooth = smooth };
00407                             areas.Add(area);
00408                             lines.Add(line);
00409                         }
00410                         else
00411                         {
00412                             Area<IReadOnlyList<double>> area = new
      Area<IReadOnlyList<double>>(groupedData[i], x => new double[] { baselines[index - 1][x[1]], x[1] },
      coordinateSystem) { PresentationAttributes = new
      PlotElementPresentationAttributes(dataPresentationAttributes[i % dataPresentationAttributes.Count]) {
      Stroke = null }, Smooth = smooth };
00413                             DataLine<IReadOnlyList<double>> line = new
      DataLine<IReadOnlyList<double>>(groupedData[i], coordinateSystem) { PresentationAttributes = new
      PlotElementPresentationAttributes(dataPresentationAttributes[i % dataPresentationAttributes.Count]) {
      Fill = null }, Smooth = smooth };
00414                             areas.Add(area);
00415                             lines.Add(line);
00416                         }
00417                     }
00418
00419                 }
00420
00421                 TextLabel<IReadOnlyList<double>> titleLabel = new
      TextLabel<IReadOnlyList<double>>(title,
      coordinateSystem.ToDataCoordinates((coordinateSystem.ToPlotCoordinates(marginTopLeft) +
      coordinateSystem.ToPlotCoordinates(marginTopRight)) * 0.5 + new Point(0, -10)), coordinateSystem) {
      Baseline = TextBaselines.Bottom, PresentationAttributes = titlePresentationAttributes };
00422
00423                 Plot tbr = new Plot();
00424                 tbr.AddPlotElements(xGrid, yGrid, xAxis, yAxis, xTicks, yTicks, xLabels, yLabels,
      xTitle, yTitle);
00425                 tbr.AddPlotElements(areas);
00426                 tbr.AddPlotElements(lines);
00427                 tbr.AddPlotElement(titleLabel);
00428
00429                 return tbr;
00430             }
00431         }
00432     }
00433 }
```

8.44 Plot.BarChart.cs

```
00001 /*
```

```
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021
00022 namespace VectSharp.Plots
00023 {
00024     partial class Plot
00025     {
00026         partial class Create
00027         {
00028 /// <summary>
00029 /// Create a new bar chart.
00030 /// </summary>
00031 /// <param name="data">The data to plot.</param>
00032 /// <param name="vertical">If this is <see langword="true"/> (the default), the bars go from the X
      axis up to the sampled values.  If this is <see langword="false"/>, the bars go from the Y axis to the
      sampled values.</param>
00033 /// <param name="margin">Spacing between consecutive bars.  This should be a value between 0 and
      1.</param>
00034 /// <param name="width">The width of the plot.</param>
00035 /// <param name="height">The height of the plot.</param>
00036 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00037 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00038 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00039 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00040 /// <param name="xAxisTitle">Title for the X axis.</param>
00041 /// <param name="yAxisTitle">Title for the Y axis.</param>
00042 /// <param name="title">Title for the plot.</param>
00043 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00044 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00045 /// <param name="dataPresentationAttributes">Presentation attributes for the plotted data.</param>
00046 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00047 /// <returns>A <see cref="Plot"/> containing the bar chart.</returns>
00048             public static Plot BarChart(IReadOnlyList<double> data, bool vertical = true, double
      margin = 0.25, double width = 350, double height = 250,
00049                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00050                 double axisArrowSize = 3,
00051                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00052                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00053                 string xAxisTitle = null,
00054                 string yAxisTitle = null,
00055                 string title = null,
00056                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00057                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00058                 PlotElementPresentationAttributes dataPresentationAttributes = null,
00059                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00060             {
00061                 double[][] changedData = new double[data.Count][];
00062
00063                 if (vertical)
00064                 {
00065                     for (int i = 0; i < data.Count; i++)
00066                     {
00067                         changedData[i] = new double[] { i, data[i] };
00068                     }
00069                 }
00070                 else
00071                 {
00072                     for (int i = 0; i < data.Count; i++)
00073                     {
00074                         changedData[i] = new double[] { data[i], i };
00075                     }
00076                 }
00077
00078                 Plot barChart = Histogram(changedData, vertical, margin, width, height,
      axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, dataPresentationAttributes, coordinateSystem);
00079
00080                 if (vertical)
```

```
00081                 {
00082                     ((ContinuousAxis)barChart.PlotElements[1]).ArrowSize = 0;
00083                 }
00084                 else
00085                 {
00086                     ((ContinuousAxis)barChart.PlotElements[2]).ArrowSize = 0;
00087                 }
00088 
00089                 Plot tbr = new Plot();
00090 
00091                 for (int i = 0; i < barChart.PlotElements.Count; i++)
00092                 {
00093                     if ((vertical && i != 5 && i != 3 && i != 1) || (!vertical && i != 6 && i != 4 &&
      i != 2))
00094                     {
00095                         tbr.AddPlotElement(barChart.PlotElements[i]);
00096                     }
00097                 }
00098 
00099                 return tbr;
00100             }
00101 
00102 /// <summary>
00103 /// Create a new bar chart.
00104 /// </summary>
00105 /// <param name="data">The data to plot.</param>
00106 /// <param name="vertical">If this is <see langword="true"/> (the default), the bars go from the X
      axis up to the sampled values.  If this is <see langword="false"/>, the bars go from the Y axis to the
      sampled values.</param>
00107 /// <param name="margin">Spacing between consecutive bars.  This should be a value between 0 and
      1.</param>
00108 /// <param name="width">The width of the plot.</param>
00109 /// <param name="height">The height of the plot.</param>
00110 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00111 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00112 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00113 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00114 /// <param name="xAxisTitle">Title for the X axis.</param>
00115 /// <param name="yAxisTitle">Title for the Y axis.</param>
00116 /// <param name="title">Title for the plot.</param>
00117 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00118 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00119 /// <param name="dataPresentationAttributes">Presentation attributes for the plotted data.</param>
00120 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00121 /// <returns>A <see cref="Plot"/> containing the bar chart.</returns>
00122             public static Plot BarChart<T>(IReadOnlyList<(T, double)> data, bool vertical = true,
      double margin = 0.25, double width = 350, double height = 250,
00123                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00124                 double axisArrowSize = 3,
00125                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00126                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00127                 string xAxisTitle = null,
00128                 string yAxisTitle = null,
00129                 string title = null,
00130                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00131                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00132                 PlotElementPresentationAttributes dataPresentationAttributes = null,
00133                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00134             {
00135                 return BarChart(data, double.NaN, double.NaN, vertical, margin, width, height,
      axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, dataPresentationAttributes, coordinateSystem);
00136             }
00137 
00138 
00139             private static Plot BarChart<T>(IReadOnlyList<(T, double)> data, double overrideMax,
      double overrideMin, bool vertical = true, double margin = 0.25, double width = 350, double height =
      250,
00140                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00141                 double axisArrowSize = 3,
00142                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00143                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00144                 string xAxisTitle = null,
00145                 string yAxisTitle = null,
00146                 string title = null,
00147                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00148                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00149                 PlotElementPresentationAttributes dataPresentationAttributes = null,
00150                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00151             {
00152                 double[][] changedData = new double[data.Count][];
00153 
00154                 if (vertical)
00155                 {
00156                     for (int i = 0; i < data.Count; i++)
```

```
00157                     {
00158                         changedData[i] = new double[] { i, data[i].Item2 };
00159                     }
00160                 }
00161                 else
00162                 {
00163                     for (int i = 0; i < data.Count; i++)
00164                     {
00165                         changedData[i] = new double[] { data[i].Item2, i };
00166                     }
00167                 }
00168 
00169                 Plot barChart = Histogram(changedData, overrideMax, overrideMin, vertical, margin,
      width, height, axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, dataPresentationAttributes, coordinateSystem);
00170 
00171                 if (vertical)
00172                 {
00173                     ((ContinuousAxis)barChart.PlotElements[1]).ArrowSize = 0;
00174                 }
00175                 else
00176                 {
00177                     ((ContinuousAxis)barChart.PlotElements[2]).ArrowSize = 0;
00178                 }
00179 
00180                 Plot tbr = new Plot();
00181 
00182                 double yLabelsWidth = 0;
00183                 double xLabelsHeight = 0;
00184 
00185                 for (int i = 0; i < barChart.PlotElements.Count; i++)
00186                 {
00187                     if ((vertical && i != 5) || (!vertical && i != 6))
00188                     {
00189                         tbr.AddPlotElement(barChart.PlotElements[i]);
00190                     }
00191                     else
00192                     {
00193                         if (vertical)
00194                         {
00195                             ContinuousAxis xAxis = barChart.GetFirst<ContinuousAxis>();
00196 
00197                             IReadOnlyList<double>[] labelData = new IReadOnlyList<double>[data.Count];
00198 
00199                             for (int j = 0; j < data.Count; j++)
00200                             {
00201                                 labelData[j] = new double[] { j, xAxis.StartPoint[1] * (1 - (double)j
      / (data.Count - 1)) + xAxis.EndPoint[1] * j / (data.Count - 1) };
00202                             }
00203 
00204                             DataLabels<IReadOnlyList<double» xLabels = new
      Plots.DataLabels<IReadOnlyList<double»(labelData,
      barChart.GetFirst<ICoordinateSystem<IReadOnlyList<double»>()) { Baseline = TextBaselines.Top, Margin =
      (a, b) => new Point(0, 10), Label = (index, _) => data[index].Item1 };
00205 
00206                             Graphics xLabelsSize = new Graphics();
00207                             xLabels.Plot(xLabelsSize);
00208                             xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
00209 
00210                             tbr.AddPlotElement(xLabels);
00211                         }
00212                         else
00213                         {
00214                             ContinuousAxis yAxis = barChart.GetAll<ContinuousAxis>().ElementAt(1);
00215 
00216                             IReadOnlyList<double>[] labelData = new IReadOnlyList<double>[data.Count];
00217 
00218                             for (int j = 0; j < data.Count; j++)
00219                             {
00220                                 labelData[j] = new double[] { yAxis.StartPoint[0] * (1 - (double)j /
      (data.Count - 1)) + yAxis.EndPoint[0] * j / (data.Count - 1), j };
00221                             }
00222 
00223                             DataLabels<IReadOnlyList<double» yLabels = new
      Plots.DataLabels<IReadOnlyList<double»(labelData,
      barChart.GetFirst<ICoordinateSystem<IReadOnlyList<double»>()) { Baseline = TextBaselines.Middle,
      Alignment = TextAnchors.Right, Margin = (a, b) => new Point(-10, 0), Label = (index, _) =>
      data[index].Item1 };
00224 
00225                             Graphics yLabelsSize = new Graphics();
00226                             yLabels.Plot(yLabelsSize);
00227                             yLabelsWidth = yLabelsSize.GetBounds().Size.Width;
00228 
00229                             tbr.AddPlotElement(yLabels);
00230                         }
00231                     }
```

```
00232                 }
00233
00234                 if (vertical)
00235                 {
00236                     tbr.GetAll<ContinuousAxisTitle>().ToArray()[0].Position = xLabelsHeight + 20;
00237                 }
00238                 else
00239                 {
00240                     tbr.GetAll<ContinuousAxisTitle>().ToArray()[1].Position = -20 - yLabelsWidth;
00241                 }
00242
00243                 return tbr;
00244             }
00245
00246 /// <summary>
00247 /// Create a new stacked bar chart.
00248 /// </summary>
00249 /// <param name="data">The data to plot.</param>
00250 /// <param name="vertical">If this is <see langword="true"/> (the default), the bars go from the X
      axis up to the sampled values.  If this is <see langword="false"/>, the bars go from the Y axis to the
      sampled values.</param>
00251 /// <param name="margin">Spacing between consecutive bars.  This should be a value between 0 and
      1.</param>
00252 /// <param name="width">The width of the plot.</param>
00253 /// <param name="height">The height of the plot.</param>
00254 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00255 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00256 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00257 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00258 /// <param name="xAxisTitle">Title for the X axis.</param>
00259 /// <param name="yAxisTitle">Title for the Y axis.</param>
00260 /// <param name="title">Title for the plot.</param>
00261 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00262 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00263 /// <param name="dataPresentationAttributes">Presentation attributes for the plotted data.</param>
00264 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00265 /// <returns>A <see cref="Plot"/> containing the stacked bar chart.</returns>
00266             public static Plot StackedBarChart(IReadOnlyList<IReadOnlyList<double» data, bool vertical
      = true, double margin = 0.25, double width = 350, double height = 250,
00267                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00268                 double axisArrowSize = 3,
00269                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00270                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00271                 string xAxisTitle = null,
00272                 string yAxisTitle = null,
00273                 string title = null,
00274                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00275                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00276                 IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
00277                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00278             {
00279                 double[][] changedData = new double[data.Count][];
00280
00281                 if (vertical)
00282                 {
00283                     for (int i = 0; i < data.Count; i++)
00284                     {
00285                         changedData[i] = new double[data[i].Count + 1];
00286
00287                         changedData[i][0] = i;
00288
00289                         for (int j = 0; j < data[i].Count; j++)
00290                         {
00291                             changedData[i][j + 1] = data[i][j];
00292                         }
00293                     }
00294                 }
00295                 else
00296                 {
00297                     for (int i = 0; i < data.Count; i++)
00298                     {
00299                         changedData[i] = new double[data[i].Count + 1];
00300
00301                         changedData[i][1] = i;
00302
00303                         changedData[i][0] = data[i][0];
00304
00305                         for (int j = 1; j < data[i].Count; j++)
00306                         {
00307                             changedData[i][j + 1] = data[i][j];
00308                         }
00309                     }
00310                 }
00311
00312                 Plot barChart = StackedHistogram(changedData, vertical, margin, width, height,
      axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
```

```
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, dataPresentationAttributes, coordinateSystem);
00313
00314                 if (vertical)
00315                 {
00316                     ((ContinuousAxis)barChart.PlotElements[1]).ArrowSize = 0;
00317                 }
00318                 else
00319                 {
00320                     ((ContinuousAxis)barChart.PlotElements[2]).ArrowSize = 0;
00321                 }
00322
00323                 Plot tbr = new Plot();
00324
00325                 for (int i = 0; i < barChart.PlotElements.Count; i++)
00326                 {
00327                     if ((vertical && i != 5 && i != 3 && i != 1) || (!vertical && i != 6 && i != 4 &&
      i != 2))
00328                     {
00329                         tbr.AddPlotElement(barChart.PlotElements[i]);
00330                     }
00331
00332                 }
00333
00334                 return tbr;
00335             }
00336
00337 /// <summary>
00338 /// Create a new stacked bar chart.
00339 /// </summary>
00340 /// <param name="data">The data to plot.</param>
00341 /// <param name="vertical">If this is <see langword="true"/> (the default), the bars go from the X
      axis up to the sampled values.  If this is <see langword="false"/>, the bars go from the Y axis to the
      sampled values.</param>
00342 /// <param name="margin">Spacing between consecutive bars.  This should be a value between 0 and
      1.</param>
00343 /// <param name="width">The width of the plot.</param>
00344 /// <param name="height">The height of the plot.</param>
00345 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00346 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00347 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00348 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00349 /// <param name="xAxisTitle">Title for the X axis.</param>
00350 /// <param name="yAxisTitle">Title for the Y axis.</param>
00351 /// <param name="title">Title for the plot.</param>
00352 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00353 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00354 /// <param name="dataPresentationAttributes">Presentation attributes for the plotted data.</param>
00355 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00356 /// <returns>A <see cref="Plot"/> containing the stacked bar chart.</returns>
00357             public static Plot StackedBarChart<T>(IReadOnlyList<(T, IReadOnlyList<double>)> data, bool
      vertical = true, double margin = 0.25, double width = 350, double height = 250,
00358                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00359                 double axisArrowSize = 3,
00360                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00361                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00362                 string xAxisTitle = null,
00363                 string yAxisTitle = null,
00364                 string title = null,
00365                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00366                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00367                 IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
00368                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00369             {
00370                 double[][] changedData = new double[data.Count][];
00371
00372                 if (vertical)
00373                 {
00374                     for (int i = 0; i < data.Count; i++)
00375                     {
00376                         changedData[i] = new double[data[i].Item2.Count + 1];
00377
00378                         changedData[i][0] = i;
00379
00380                         for (int j = 0; j < data[i].Item2.Count; j++)
00381                         {
00382                             changedData[i][j + 1] = data[i].Item2[j];
00383                         }
00384                     }
00385                 }
00386                 else
00387                 {
00388                     for (int i = 0; i < data.Count; i++)
00389                     {
00390                         changedData[i] = new double[data[i].Item2.Count + 1];
00391
```

```
00392                           changedData[i][1] = i;
00393
00394                           changedData[i][0] = data[i].Item2[0];
00395
00396                           for (int j = 1; j < data[i].Item2.Count; j++)
00397                           {
00398                               changedData[i][j + 1] = data[i].Item2[j];
00399                           }
00400                       }
00401                   }
00402
00403                   Plot barChart = StackedHistogram(changedData, vertical, margin, width, height,
      axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, dataPresentationAttributes, coordinateSystem);
00404
00405                   if (vertical)
00406                   {
00407                       ((ContinuousAxis)barChart.PlotElements[1]).ArrowSize = 0;
00408                   }
00409                   else
00410                   {
00411                       ((ContinuousAxis)barChart.PlotElements[2]).ArrowSize = 0;
00412                   }
00413
00414                   Plot tbr = new Plot();
00415
00416                   double xLabelsHeight = 0;
00417                   double yLabelsWidth = 0;
00418
00419                   for (int i = 0; i < barChart.PlotElements.Count; i++)
00420                   {
00421                       if ((vertical && i != 5) || (!vertical && i != 6))
00422                       {
00423                           tbr.AddPlotElement(barChart.PlotElements[i]);
00424                       }
00425                       else
00426                       {
00427                           if (vertical)
00428                           {
00429                               ContinuousAxis xAxis = barChart.GetFirst<ContinuousAxis>();
00430
00431                               IReadOnlyList<double>[] labelData = new IReadOnlyList<double>[data.Count];
00432
00433                               for (int j = 0; j < data.Count; j++)
00434                               {
00435                                   if (data.Count > 1)
00436                                   {
00437                                       labelData[j] = new double[] { j, xAxis.StartPoint[1] * (1 -
      (double)j / (data.Count - 1)) + xAxis.EndPoint[1] * j / (data.Count - 1) };
00438                                   }
00439                                   else
00440                                   {
00441                                       labelData[j] = new double[] { j, xAxis.StartPoint[1] * 0.5 +
      xAxis.EndPoint[1] * 0.5 };
00442                                   }
00443                               }
00444
00445                               DataLabels<IReadOnlyList<double» xLabels = new
      Plots.DataLabels<IReadOnlyList<double»(labelData,
      barChart.GetFirst<ICoordinateSystem<IReadOnlyList<double»>()) { Baseline = TextBaselines.Top, Margin =
      (a, b) => new Point(0, 10), Label = (index, _) => data[index].Item1 };
00446
00447                               Graphics xLabelsSize = new Graphics();
00448                               xLabels.Plot(xLabelsSize);
00449                               xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
00450
00451
00452                               tbr.AddPlotElement(xLabels);
00453                           }
00454                           else
00455                           {
00456                               ContinuousAxis yAxis = barChart.GetAll<ContinuousAxis>().ElementAt(1);
00457
00458                               IReadOnlyList<double>[] labelData = new IReadOnlyList<double>[data.Count];
00459
00460                               for (int j = 0; j < data.Count; j++)
00461                               {
00462                                   if (data.Count > 1)
00463                                   {
00464                                       labelData[j] = new double[] { yAxis.StartPoint[0] * (1 - (double)j
      / (data.Count - 1)) + yAxis.EndPoint[0] * j / (data.Count - 1), j };
00465                                   }
00466                                   else
00467                                   {
00468                                       labelData[j] = new double[] { yAxis.StartPoint[0] * 0.5 +
      yAxis.EndPoint[0] * 0.5, j };
```

```
00469                                 }
00470                             }
00471 
00472                             DataLabels<IReadOnlyList<double» yLabels = new
      Plots.DataLabels<IReadOnlyList<double»(labelData,
      barChart.GetFirst<ICoordinateSystem<IReadOnlyList<double»>()) { Baseline = TextBaselines.Middle,
      Alignment = TextAnchors.Right, Margin = (a, b) => new Point(-10, 0), Label = (index, _) =>
      data[index].Item1 };
00473 
00474                             Graphics yLabelsSize = new Graphics();
00475                             yLabels.Plot(yLabelsSize);
00476                             yLabelsWidth = yLabelsSize.GetBounds().Size.Width;
00477 
00478                             tbr.AddPlotElement(yLabels);
00479                         }
00480                     }
00481                 }
00482 
00483                 if (vertical)
00484                 {
00485                     tbr.GetAll<ContinuousAxisTitle>().ToArray()[0].Position = xLabelsHeight + 20;
00486                 }
00487                 else
00488                 {
00489                     tbr.GetAll<ContinuousAxisTitle>().ToArray()[1].Position = -20 - yLabelsWidth;
00490                 }
00491 
00492 
00493                 return tbr;
00494             }
00495 
00496 /// <summary>
00497 /// Create a new clustered bar chart.
00498 /// </summary>
00499 /// <param name="data">The data to plot.</param>
00500 /// <param name="vertical">If this is <see langword="true"/> (the default), the bars go from the X
      axis up to the sampled values.  If this is <see langword="false"/>, the bars go from the Y axis to the
      sampled values.</param>
00501 /// <param name="interClusterMargin">Spacing between consecutive bar clusters.  This should be a value
      between 0 and 1.</param>
00502 /// <param name="intraClusterMargin">Spacing between consecutive bars within the same clusters.  This
      should be a value between 0 and 1.</param>
00503 /// <param name="width">The width of the plot.</param>
00504 /// <param name="height">The height of the plot.</param>
00505 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00506 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00507 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00508 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00509 /// <param name="xAxisTitle">Title for the X axis.</param>
00510 /// <param name="yAxisTitle">Title for the Y axis.</param>
00511 /// <param name="title">Title for the plot.</param>
00512 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00513 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00514 /// <param name="dataPresentationAttributes">Presentation attributes for the plotted data.</param>
00515 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00516 /// <returns>A <see cref="Plot"/> containing the bar chart.</returns>
00517             public static Plot ClusteredBarChart(IReadOnlyList<IReadOnlyList<double» data, bool
      vertical = true, double interClusterMargin = 0.25, double intraClusterMargin = 0, double width = 350,
      double height = 250,
00518                     PlotElementPresentationAttributes axisPresentationAttributes = null,
00519                     double axisArrowSize = 3,
00520                     PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00521                     PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00522                     string xAxisTitle = null,
00523                     string yAxisTitle = null,
00524                     string title = null,
00525                     PlotElementPresentationAttributes titlePresentationAttributes = null,
00526                     PlotElementPresentationAttributes gridPresentationAttributes = null,
00527                     IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes =
      null,
00528                     IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00529             {
00530                 if (axisPresentationAttributes == null)
00531                 {
00532                     axisPresentationAttributes = new PlotElementPresentationAttributes();
00533                 }
00534 
00535                 if (gridPresentationAttributes == null)
00536                 {
00537                     gridPresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      new SolidColourBrush(Colour.FromRgb(220, 220, 220)) };
00538                 }
00539 
00540                 if (axisLabelPresentationAttributes == null)
00541                 {
00542                     axisLabelPresentationAttributes = new PlotElementPresentationAttributes() { Stroke
```

```
= null };
00543                 }
00544
00545                 if (axisTitlePresentationAttributes == null)
00546                 {
00547                     axisTitlePresentationAttributes = new PlotElementPresentationAttributes() { Font =
new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 14), Stroke =
null };
00548                 }
00549
00550                 if (dataPresentationAttributes == null)
00551                 {
00552                     dataPresentationAttributes = new PlotElementPresentationAttributes[] { new
PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(0,
114, 178)) },
00553                                                                                            new
PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(213,
94, 0)) },
00554                                                                                            new
PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(204,
121, 167)) },
00555                                                                                            new
PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(230,
159, 0)) },
00556                                                                                            new
PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(86,
180, 233)) },
00557                                                                                            new
PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(0,
158, 115)) },
00558                                                                                            new
PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(240,
228, 66)) } };
00559                 }
00560
00561                 double baselineValue = 0;
00562
00563                 if (coordinateSystem is LogarithmicCoordinateSystem2D || (!vertical &&
coordinateSystem is LinLogCoordinateSystem2D) || (vertical && coordinateSystem is
LogLinCoordinateSystem2D))
00564                 {
00565                     baselineValue = 1;
00566                 }
00567
00568                 ClusteredBars bars;
00569
00570                 if (baselineValue == 0)
00571                 {
00572                     bars = new ClusteredBars(data, coordinateSystem, vertical) {
PresentationAttributes = dataPresentationAttributes, InterClusterMargin = interClusterMargin,
IntraClusterMargin = intraClusterMargin };
00573                 }
00574                 else
00575                 {
00576                     bars = new ClusteredBars(data, new Comparison<IReadOnlyList<double»(vertical ?
new Func<IReadOnlyList<double>, IReadOnlyList<double>, int>((x, y) => Math.Sign(x[0] - y[0])) :  (x,
y) => Math.Sign(x[1] - y[1])), vertical ?  new Func<IReadOnlyList<double>, IReadOnlyList<double»(pt =>
new double[] { pt[0], baselineValue }) :  pt => new double[] { baselineValue, pt[1] },
coordinateSystem) { PresentationAttributes = dataPresentationAttributes, InterClusterMargin =
interClusterMargin, IntraClusterMargin = intraClusterMargin };
00577                 }
00578
00579                 double minX, minY, maxX, maxY, rangeX, rangeY;
00580
00581                 {
00582                     minX = double.MaxValue;
00583                     minY = double.MaxValue;
00584
00585                     maxX = double.MinValue;
00586                     maxY = double.MinValue;
00587
00588                     for (int i = 0; i < data.Count; i++)
00589                     {
00590                         minX = Math.Min(minX, data[i][0]);
00591                         minY = Math.Min(minY, data[i][1]);
00592
00593                         maxX = Math.Max(maxX, data[i][0]);
00594                         maxY = Math.Max(maxY, data[i][1]);
00595
00596                         if (vertical)
00597                         {
00598                             for (int j = 2; j < data[i].Count; j++)
00599                             {
00600                                 minY = Math.Min(minY, data[i][j]);
00601                                 maxY = Math.Max(maxY, data[i][j]);
00602                             }
00603                         }
```

```
00604                         else
00605                         {
00606                             for (int j = 2; j < data[i].Count; j++)
00607                             {
00608                                 minX = Math.Min(minX, data[i][j]);
00609                                 maxX = Math.Max(maxX, data[i][j]);
00610                             }
00611                         }
00612                     }
00613 
00614                     rangeX = maxX - minX;
00615                     rangeY = maxY - minY;
00616                 }
00617 
00618                 double dataMinX = minX;
00619                 double dataMinY = minY;
00620                 double dataMaxX = maxX;
00621                 double dataMaxY = maxY;
00622 
00623                 if (coordinateSystem == null)
00624                 {
00625                     LinearCoordinateSystem2D linCoords = new LinearCoordinateSystem2D(data, width,
      height);
00626 
00627                     if (vertical)
00628                     {
00629                         minY = Math.Min(minY, 0);
00630                         maxY = Math.Max(maxY, 0);
00631                         rangeY = maxY - minY;
00632 
00633                         linCoords.MinY = minY - rangeY * 0.1;
00634                         linCoords.MaxY = maxY + rangeY * 0.1;
00635 
00636 
00637                         if (bars.Data.Count >= 2)
00638                         {
00639                             IReadOnlyList<double> item0 = bars.Data.ElementAt(0);
00640                             IReadOnlyList<double> item1 = bars.Data.ElementAt(1);
00641 
00642                             IReadOnlyList<double> itemN = bars.Data.ElementAt(bars.Data.Count - 1);
00643                             IReadOnlyList<double> itemN1 = bars.Data.ElementAt(bars.Data.Count - 2);
00644 
00645                             minX = Math.Min(minX, 1.5 * item0[0] - item1[0] * 0.5);
00646                             maxX = Math.Max(maxX, 1.5 * itemN[0] - itemN1[0] * 0.5);
00647 
00648                             rangeX = maxX - minX;
00649                             linCoords.MinX = minX;
00650                             linCoords.MaxX = maxX;
00651                         }
00652                         else if (bars.Data.Count == 1)
00653                         {
00654                             minX = Math.Min(minX, -0.5);
00655                             maxX = Math.Max(maxX, 0.5);
00656                             dataMinX = minX;
00657                             dataMaxX = maxX;
00658 
00659                             rangeX = maxX - minX;
00660                             linCoords.MinX = minX;
00661                             linCoords.MaxX = maxX;
00662                         }
00663                     }
00664                     else
00665                     {
00666                         minX = Math.Min(minX, 0);
00667                         maxX = Math.Max(maxX, 0);
00668                         rangeX = maxX - minX;
00669 
00670                         linCoords.MinX = minX - rangeX * 0.1;
00671                         linCoords.MaxX = maxX + rangeX * 0.1;
00672 
00673 
00674                         if (bars.Data.Count >= 2)
00675                         {
00676                             IReadOnlyList<double> item0 = bars.Data.ElementAt(0);
00677                             IReadOnlyList<double> item1 = bars.Data.ElementAt(1);
00678 
00679                             IReadOnlyList<double> itemN = bars.Data.ElementAt(bars.Data.Count - 1);
00680                             IReadOnlyList<double> itemN1 = bars.Data.ElementAt(bars.Data.Count - 2);
00681 
00682                             minY = Math.Min(minY, 1.5 * item0[1] - item1[1] * 0.5);
00683                             maxY = Math.Max(maxY, 1.5 * itemN[1] - itemN1[1] * 0.5);
00684 
00685                             rangeY = maxY - minY;
00686                             linCoords.MinY = minY;
00687                             linCoords.MaxY = maxY;
00688                         }
00689                         else if (bars.Data.Count == 1)
```

```
00690                         {
00691                             minY = Math.Min(minY, -0.5);
00692                             maxY = Math.Max(maxY, 0.5);
00693                             dataMinY = minY;
00694                             dataMaxY = maxY;
00695 
00696                             rangeY = maxY - minY;
00697                             linCoords.MinY = minY;
00698                             linCoords.MaxY = maxY;
00699                         }
00700                     }
00701 
00702                     coordinateSystem = linCoords;
00703                     bars.CoordinateSystem = coordinateSystem;
00704                 }
00705                 else
00706                 {
00707                     if (vertical)
00708                     {
00709                         minY = Math.Min(minY, baselineValue);
00710                         rangeY = maxY - minY;
00711 
00712                         if (bars.Data.Count >= 2)
00713                         {
00714                             IReadOnlyList<double> item0 = bars.Data.ElementAt(0);
00715                             IReadOnlyList<double> item1 = bars.Data.ElementAt(1);
00716 
00717                             IReadOnlyList<double> itemN = bars.Data.ElementAt(bars.Data.Count - 1);
00718                             IReadOnlyList<double> itemN1 = bars.Data.ElementAt(bars.Data.Count - 2);
00719 
00720                             minX = Math.Min(minX, 1.5 * item0[0] - item1[0] * 0.5);
00721                             maxX = Math.Max(maxX, 1.5 * itemN[0] - itemN1[0] * 0.5);
00722 
00723                             rangeX = maxX - minX;
00724                         }
00725                         else if (bars.Data.Count == 1)
00726                         {
00727                             minX = Math.Min(minX, -0.5);
00728                             maxX = Math.Max(maxX, 0.5);
00729                             dataMinX = minX;
00730                             dataMaxX = maxX;
00731 
00732                             rangeX = maxX - minX;
00733                         }
00734                     }
00735                     else
00736                     {
00737                         minX = Math.Min(minX, baselineValue);
00738                         rangeX = maxX - minX;
00739 
00740                         if (bars.Data.Count >= 2)
00741                         {
00742                             IReadOnlyList<double> item0 = bars.Data.ElementAt(0);
00743                             IReadOnlyList<double> item1 = bars.Data.ElementAt(1);
00744 
00745                             IReadOnlyList<double> itemN = bars.Data.ElementAt(bars.Data.Count - 1);
00746                             IReadOnlyList<double> itemN1 = bars.Data.ElementAt(bars.Data.Count - 2);
00747 
00748                             minY = Math.Min(minY, 1.5 * item0[1] - item1[1] * 0.5);
00749                             maxY = Math.Max(maxY, 1.5 * itemN[1] - itemN1[1] * 0.5);
00750 
00751                             rangeY = maxY - minY;
00752                         }
00753                         else if (bars.Data.Count == 1)
00754                         {
00755                             minY = Math.Min(minY, -0.5);
00756                             maxY = Math.Max(maxY, 0.5);
00757                             dataMinY = minY;
00758                             dataMaxY = maxY;
00759 
00760                             rangeY = maxY - minY;
00761                         }
00762                     }
00763                 }
00764 
00765 
00766 
00767                 if (titlePresentationAttributes == null)
00768                 {
00769                     titlePresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 18)
      };
00770                 }
00771 
00772                 Point topLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, maxY });
00773                 Point topRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, maxY });
00774                 Point bottomRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, minY });
```

```
00775                 Point bottomLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, minY });
00776
00777                 double[] marginTopLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00778                 double[] marginTopRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00779                 double[] marginBottomRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00780                 double[] marginBottomLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00781
00782                 double[] p1 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00783                 double[] p2 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00784                 double[] p3 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00785                 double[] p4 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00786
00787
00788                 double[] p5 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00789                 double[] p6 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00790                 double[] p7 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00791                 double[] p8 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00792
00793                 if (vertical)
00794                 {
00795                     Point p1p = new Point(Math.Min(Math.Min(topLeft.X, topRight.X),
      Math.Min(bottomLeft.X, bottomRight.X)), Math.Min(Math.Min(topLeft.Y, topRight.Y),
      Math.Min(bottomLeft.Y, bottomRight.Y)) - 10);
00796                     Point p1p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMinX, dataMaxY
      });
00797
00798                     Point p2p = new Point(Math.Max(Math.Max(topLeft.X, topRight.X),
      Math.Max(bottomLeft.X, bottomRight.X)), Math.Min(Math.Min(topLeft.Y, topRight.Y),
      Math.Min(bottomLeft.Y, bottomRight.Y)) - 10);
00799                     Point p2p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMaxX, dataMaxY
      });
00800
00801                     Point p3p = new Point(Math.Min(Math.Min(topLeft.X, topRight.X),
      Math.Min(bottomLeft.X, bottomRight.X)), Math.Max(Math.Max(topLeft.Y, topRight.Y),
      Math.Max(bottomLeft.Y, bottomRight.Y)) + 10);
00802                     Point p3p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMinX, 0 });
00803
00804                     Point p4p = new Point(Math.Max(Math.Max(topLeft.X, topRight.X),
      Math.Max(bottomLeft.X, bottomRight.X)), Math.Max(Math.Max(topLeft.Y, topRight.Y),
      Math.Max(bottomLeft.Y, bottomRight.Y)) + 10);
00805                     Point p4p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMaxX, 0 });
00806
00807                     p1 = coordinateSystem.ToDataCoordinates(new Point(p1p2.X, p1p.Y));
00808                     p2 = coordinateSystem.ToDataCoordinates(new Point(p2p2.X, p2p.Y));
00809                     p3 = coordinateSystem.ToDataCoordinates(new Point(p3p2.X, p3p.Y));
00810                     p4 = coordinateSystem.ToDataCoordinates(new Point(p4p2.X, p4p.Y));
00811                 }
00812                 else
00813                 {
00814                     Point p5p = new Point(Math.Min(Math.Min(topLeft.X, topRight.X),
      Math.Min(bottomLeft.X, bottomRight.X)) - 10, Math.Min(Math.Min(topLeft.Y, topRight.Y),
      Math.Min(bottomLeft.Y, bottomRight.Y)));
00815                     Point p5p2 = coordinateSystem.ToPlotCoordinates(new double[] { 0, dataMaxY });
00816
00817                     Point p6p = new Point(Math.Min(Math.Min(topLeft.X, topRight.X),
      Math.Min(bottomLeft.X, bottomRight.X)) - 10, Math.Max(Math.Max(topLeft.Y, topRight.Y),
      Math.Max(bottomLeft.Y, bottomRight.Y)));
00818                     Point p6p2 = coordinateSystem.ToPlotCoordinates(new double[] { 0, dataMinY });
00819
00820                     Point p7p = new Point(Math.Max(Math.Max(topLeft.X, topRight.X),
      Math.Max(bottomLeft.X, bottomRight.X)) + 10, Math.Min(Math.Min(topLeft.Y, topRight.Y),
      Math.Min(bottomLeft.Y, bottomRight.Y)));
00821                     Point p7p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMaxX, dataMaxY
```

```
    });
00822
00823                   Point p8p = new Point(Math.Max(Math.Max(topLeft.X, topRight.X),
    Math.Max(bottomLeft.X, bottomRight.X)) + 10, Math.Max(Math.Max(topLeft.Y, topRight.Y),
    Math.Max(bottomLeft.Y, bottomRight.Y)));
00824                   Point p8p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMinX, dataMinY
    });
00825
00826                   p5 = coordinateSystem.ToDataCoordinates(new Point(p5p.X, p5p2.Y));
00827                   p6 = coordinateSystem.ToDataCoordinates(new Point(p6p.X, p6p2.Y));
00828                   p7 = coordinateSystem.ToDataCoordinates(new Point(p7p2.X, p7p.Y));
00829                   p8 = coordinateSystem.ToDataCoordinates(new Point(p8p2.X, p8p.Y));
00830               }
00831
00832               Grid grid;
00833
00834               if (vertical)
00835               {
00836                   grid = new Grid(p5, p6, p7, p8, coordinateSystem) { IntervalCount = 5,
    PresentationAttributes = gridPresentationAttributes };
00837               }
00838               else
00839               {
00840                   grid = new Grid(p1, p2, p3, p4, coordinateSystem) { IntervalCount = 5,
    PresentationAttributes = gridPresentationAttributes };
00841               }
00842
00843               ContinuousAxis xAxis = new ContinuousAxis(marginBottomLeft[0] < marginBottomRight[0] ?
    marginBottomLeft :  marginBottomRight, marginBottomLeft[0] < marginBottomRight[0] ?  marginBottomRight
    :  vertical ?  marginBottomLeft :
    coordinateSystem.ToDataCoordinates(coordinateSystem.ToPlotCoordinates(marginBottomLeft) + new
    Point(-axisArrowSize - 7, 0)), coordinateSystem) { PresentationAttributes =
    axisPresentationAttributes, ArrowSize = axisArrowSize };
00844               ContinuousAxis yAxis = new ContinuousAxis(marginBottomLeft[1] < marginTopLeft[1] ?
    marginBottomLeft :  marginTopLeft, marginBottomLeft[1] < marginTopLeft[1] ?  marginTopLeft :
    !vertical ?  marginBottomLeft :
    coordinateSystem.ToDataCoordinates(coordinateSystem.ToPlotCoordinates(marginBottomLeft) + new Point(0,
    axisArrowSize + 7)), coordinateSystem) { PresentationAttributes = axisPresentationAttributes,
    ArrowSize = axisArrowSize };
00845
00846               int every = (int)Math.Ceiling((double)data.Count / 5);
00847               int shift = (data.Count - every * (data.Count / every - 1) - 1) / 2;
00848
00849               ContinuousAxisTicks xTicks;
00850               ContinuousAxisTicks yTicks;
00851
00852               if (vertical)
00853               {
00854                   xTicks = new ContinuousAxisTicks(p3, p4, coordinateSystem) {
    PresentationAttributes = axisPresentationAttributes, IntervalCount = data.Count - 1, SizeAbove = i =>
    (i - shift) % every == 0 ?  3 :  2, SizeBelow = i => (i - shift) % every == 0 ?  3 :  2 };
00855                   yTicks = new ContinuousAxisTicks(p6, p5, coordinateSystem) {
    PresentationAttributes = axisPresentationAttributes };
00856               }
00857               else
00858               {
00859                   xTicks = new ContinuousAxisTicks(p3, p4, coordinateSystem) {
    PresentationAttributes = axisPresentationAttributes };
00860                   yTicks = new ContinuousAxisTicks(p6, p5, coordinateSystem) {
    PresentationAttributes = axisPresentationAttributes, IntervalCount = data.Count - 1, SizeAbove = i =>
    (i - shift) % every == 0 ?  3 :  2, SizeBelow = i => (i - shift) % every == 0 ?  3 :  2 };
00861               }
00862
00863               ContinuousAxisLabels xLabels;
00864               ContinuousAxisLabels yLabels;
00865
00866               if (vertical)
00867               {
00868                   xLabels = new ContinuousAxisLabels(p3, p4, coordinateSystem) {
    PresentationAttributes = axisLabelPresentationAttributes, Alignment = TextAnchors.Center, Baseline =
    TextBaselines.Top, Rotation = 0, IntervalCount = data.Count - 1 };
00869                   yLabels = new ContinuousAxisLabels(p6, p5, coordinateSystem) {
    PresentationAttributes = axisLabelPresentationAttributes, Position = _ => -10, Alignment =
    TextAnchors.Right, Rotation = 0, IntervalCount = 5 };
00870
00871                   Func<IReadOnlyList<double>, int, IEnumerable<FormattedText» originalFormatter =
    xLabels.TextFormat;
00872                   xLabels.TextFormat = (x, i) => (i - shift) % every == 0 ?  originalFormatter(x, i)
    :  null;
00873               }
00874               else
00875               {
00876                   xLabels = new ContinuousAxisLabels(p3, p4, coordinateSystem) {
    PresentationAttributes = axisLabelPresentationAttributes, Alignment = TextAnchors.Center, Baseline =
    TextBaselines.Top, Rotation = 0, IntervalCount = 5 };
00877                   yLabels = new ContinuousAxisLabels(p6, p5, coordinateSystem) {
    PresentationAttributes = axisLabelPresentationAttributes, Position = _ => -10, Alignment =
```

```
      TextAnchors.Right, Rotation = 0, IntervalCount = data.Count - 1 };
00878
00879                     Func<IReadOnlyList<double>, int, IEnumerable<FormattedText» originalFormatter =
      yLabels.TextFormat;
00880                     yLabels.TextFormat = (x, i) => (i - shift) % every == 0 ?  originalFormatter(x, i)
      :  null;
00881                 }
00882
00883                 Graphics xLabelsSize = new Graphics();
00884                 xLabels.Plot(xLabelsSize);
00885                 double xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
00886
00887                 Graphics yLabelsSize = new Graphics();
00888                 yLabels.Plot(yLabelsSize);
00889                 double yLabelsWidth = yLabelsSize.GetBounds().Size.Width;
00890
00891                 ContinuousAxisTitle xTitle = new ContinuousAxisTitle(xAxisTitle, marginBottomLeft,
      marginBottomRight, coordinateSystem, axisTitlePresentationAttributes) { Position = xLabelsHeight + 20,
      Alignment = TextAnchors.Center };
00892                 ContinuousAxisTitle yTitle = new ContinuousAxisTitle(yAxisTitle, marginBottomLeft,
      marginTopLeft, coordinateSystem, axisTitlePresentationAttributes) { Position = -20 - yLabelsWidth,
      Baseline = TextBaselines.Bottom, Alignment = TextAnchors.Center };
00893
00894                 TextLabel<IReadOnlyList<double» titleLabel = new
      TextLabel<IReadOnlyList<double»(title,
      coordinateSystem.ToDataCoordinates((coordinateSystem.ToPlotCoordinates(marginTopLeft) +
      coordinateSystem.ToPlotCoordinates(marginTopRight)) * 0.5 + new Point(0, -10)), coordinateSystem) {
      Baseline = TextBaselines.Bottom, PresentationAttributes = titlePresentationAttributes };
00895
00896                 Plot tbr = new Plot();
00897                 tbr.AddPlotElements(grid, xAxis, yAxis, xTicks, yTicks, xLabels, yLabels, xTitle,
      yTitle, bars, titleLabel);
00898
00899                 return tbr;
00900             }
00901
00902 /// <summary>
00903 /// Create a new clustered bar chart.
00904 /// </summary>
00905 /// <param name="data">The data to plot.</param>
00906 /// <param name="vertical">If this is <see langword="true"/> (the default), the bars go from the X
      axis up to the sampled values.  If this is <see langword="false"/>, the bars go from the Y axis to the
      sampled values.</param>
00907 /// <param name="interClusterMargin">Spacing between consecutive bar clusters.  This should be a value
      between 0 and 1.</param>
00908 /// <param name="intraClusterMargin">Spacing between consecutive bars within the same clusters.  This
      should be a value between 0 and 1.</param>
00909 /// <param name="width">The width of the plot.</param>
00910 /// <param name="height">The height of the plot.</param>
00911 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00912 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00913 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00914 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00915 /// <param name="xAxisTitle">Title for the X axis.</param>
00916 /// <param name="yAxisTitle">Title for the Y axis.</param>
00917 /// <param name="title">Title for the plot.</param>
00918 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00919 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00920 /// <param name="dataPresentationAttributes">Presentation attributes for the plotted data.</param>
00921 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00922 /// <returns>A <see cref="Plot"/> containing the bar chart.</returns>
00923             public static Plot ClusteredBarChart<T>(IReadOnlyList<(T, IReadOnlyList<double>)> data,
      bool vertical = true, double interClusterMargin = 0.25, double intraClusterMargin = 0, double width =
      350, double height = 250,
00924                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00925                 double axisArrowSize = 3,
00926                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00927                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00928                 string xAxisTitle = null,
00929                 string yAxisTitle = null,
00930                 string title = null,
00931                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00932                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00933                 IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
00934                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00935             {
00936                 double[][] changedData = new double[data.Count][];
00937
00938                 if (vertical)
00939                 {
00940                     for (int i = 0; i < data.Count; i++)
00941                     {
00942                         changedData[i] = new double[data[i].Item2.Count + 1];
00943
00944                         changedData[i][0] = i;
00945
```

```
00946                         for (int j = 0; j < data[i].Item2.Count; j++)
00947                         {
00948                             changedData[i][j + 1] = data[i].Item2[j];
00949                         }
00950                     }
00951                 }
00952                 else
00953                 {
00954                     for (int i = 0; i < data.Count; i++)
00955                     {
00956                         changedData[i] = new double[data[i].Item2.Count + 1];
00957 
00958                         changedData[i][1] = i;
00959 
00960                         changedData[i][0] = data[i].Item2[0];
00961 
00962                         for (int j = 1; j < data[i].Item2.Count; j++)
00963                         {
00964                             changedData[i][j + 1] = data[i].Item2[j];
00965                         }
00966                     }
00967                 }
00968 
00969                 Plot barChart = ClusteredBarChart(changedData, vertical, interClusterMargin,
      intraClusterMargin, width, height, axisPresentationAttributes, axisArrowSize,
      axisLabelPresentationAttributes, axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title,
      titlePresentationAttributes, gridPresentationAttributes, dataPresentationAttributes,
      coordinateSystem);
00970 
00971                 if (vertical)
00972                 {
00973                     ((ContinuousAxis)barChart.PlotElements[1]).ArrowSize = 0;
00974                 }
00975                 else
00976                 {
00977                     ((ContinuousAxis)barChart.PlotElements[2]).ArrowSize = 0;
00978                 }
00979 
00980                 Plot tbr = new Plot();
00981 
00982                 double xLabelsHeight = 0;
00983                 double yLabelsWidth = 0;
00984 
00985                 for (int i = 0; i < barChart.PlotElements.Count; i++)
00986                 {
00987                     if ((vertical && i != 5) || (!vertical && i != 6))
00988                     {
00989                         tbr.AddPlotElement(barChart.PlotElements[i]);
00990                     }
00991                     else
00992                     {
00993                         if (vertical)
00994                         {
00995                             ContinuousAxis xAxis = barChart.GetFirst<ContinuousAxis>();
00996 
00997                             IReadOnlyList<double>[] labelData = new IReadOnlyList<double>[data.Count];
00998 
00999                             for (int j = 0; j < data.Count; j++)
01000                             {
01001                                 if (data.Count > 1)
01002                                 {
01003                                     labelData[j] = new double[] { j, xAxis.StartPoint[1] * (1 -
      (double)j / (data.Count - 1)) + xAxis.EndPoint[1] * j / (data.Count - 1) };
01004                                 }
01005                                 else
01006                                 {
01007                                     labelData[j] = new double[] { j, xAxis.StartPoint[1] * 0.5 +
      xAxis.EndPoint[1] * 0.5 };
01008                                 }
01009                             }
01010 
01011                             DataLabels<IReadOnlyList<double> xLabels = new
      Plots.DataLabels<IReadOnlyList<double>(labelData,
      barChart.GetFirst<ICoordinateSystem<IReadOnlyList<double>>()) { Baseline = TextBaselines.Top, Margin =
      (a, b) => new Point(0, 10), Label = (index, _) => data[index].Item1 };
01012 
01013                             Graphics xLabelsSize = new Graphics();
01014                             xLabels.Plot(xLabelsSize);
01015                             xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
01016 
01017                             tbr.AddPlotElement(xLabels);
01018                         }
01019                         else
01020                         {
01021                             ContinuousAxis yAxis = barChart.GetAll<ContinuousAxis>().ElementAt(1);
01022 
01023                             IReadOnlyList<double>[] labelData = new IReadOnlyList<double>[data.Count];
```

```
01024
01025                           for (int j = 0; j < data.Count; j++)
01026                           {
01027                               if (data.Count > 1)
01028                               {
01029                                   labelData[j] = new double[] { yAxis.StartPoint[0] * 0.5 +
      yAxis.EndPoint[0] * 0.5, j };
01030                               }
01031                               else
01032                               {
01033                                   labelData[j] = new double[] { yAxis.StartPoint[0] * 0.5 +
      yAxis.EndPoint[0] * 0.5, j };
01034                               }
01035                           }
01036
01037                           DataLabels<IReadOnlyList<double» yLabels = new
      Plots.DataLabels<IReadOnlyList<double»(labelData,
      barChart.GetFirst<ICoordinateSystem<IReadOnlyList<double»>()) { Baseline = TextBaselines.Middle,
      Alignment = TextAnchors.Right, Margin = (a, b) => new Point(-10, 0), Label = (index, _) =>
      data[index].Item1 };
01038
01039                           Graphics yLabelsSize = new Graphics();
01040                           yLabels.Plot(yLabelsSize);
01041                           yLabelsWidth = yLabelsSize.GetBounds().Size.Width;
01042
01043                           tbr.AddPlotElement(yLabels);
01044                       }
01045                   }
01046
01047               }
01048
01049               if (vertical)
01050               {
01051                   tbr.GetAll<ContinuousAxisTitle>().ToArray()[0].Position = xLabelsHeight + 20;
01052               }
01053               else
01054               {
01055                   tbr.GetAll<ContinuousAxisTitle>().ToArray()[1].Position = -20 - yLabelsWidth;
01056               }
01057
01058               return tbr;
01059           }
01060       }
01061   }
01062 }
```

8.45 Plot.BoxPlot.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using MathNet.Numerics.Statistics;
00019 using System;
00020 using System.Collections.Generic;
00021 using System.Linq;
00022
00023 namespace VectSharp.Plots
00024 {
00025     partial class Plot
00026     {
00027 /// <summary>
00028 /// Describes the types of whiskers for a box plot.
00029 /// </summary>
00030         public enum WhiskerType
00031         {
00032 /// <summary>
00033 /// The whiskers extend from the minimum sampled value to the maximum sampled value.
00034 /// </summary>
00035             FullRange,
```

```
00036
00037 /// <summary>
00038 /// The whiskers extend from (Q1 - 1.5 * IQR) to (Q3 + 1.5 * IQR).
00039 /// </summary>
00040             IQR_1_5,
00041
00042 /// <summary>
00043 /// The whiskers extend from (mean - 2 * standard deviation) to (mean + 2 * standard deviation).
00044 /// </summary>
00045             StandardDeviation
00046         }
00047
00048         partial class Create
00049         {
00050 /// <summary>
00051 /// Create a new box plot.
00052 /// </summary>
00053 /// <param name="data">The data to plot.</param>
00054 /// <param name="vertical">If this is <see langword="true"/> (the default), the boxes are parallel to
      the Y axis.  If this is <see langword="false"/>, the boxes are parallel to the X axis.</param>
00055 /// <param name="whiskerType">The type of whiskers to use in the plot.</param>
00056 /// <param name="useNotches">If this is <see langword="true"/> (the default), the box plot has
      notches.  Otherwise, it does not.</param>
00057 /// <param name="proportionalWidth">If this is <see langword="false"/> (the default), all the boxes
      have the same width.  Otherwise, the width of each box is proportional to the number of
      samples.</param>
00058 /// <param name="showOutliers">If this is <see langword="true"/> (the default), a symbol is drawn four
      outlier points that fall outside of the interval between the whiskers.</param>
00059 /// <param name="boxWidth">The width of the boxes in data space coordinates.</param>
00060 /// <param name="spacing">The spacing between consecutive boxes (ranging from 0 to 1).</param>
00061 /// <param name="dataRangeMin">If this is not <see langword="null"/>, this value is used to override
      the default minimum value for the plot.  Useful if you wish to create multiple plots with the same
      scale,
00062 /// even though the sampled values have different ranges.</param>
00063 /// <param name="dataRangeMax">If this is not <see langword="null"/>, this value is used to override
      the default maximum value for the plot.  Useful if you wish to create multiple plots with the same
      scale,
00064 /// even though the sampled values have different ranges.</param>
00065 /// <param name="width">The width of the plot.</param>
00066 /// <param name="height">The height of the plot.</param>
00067 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00068 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00069 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00070 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00071 /// <param name="xAxisTitle">Title for the X axis.</param>
00072 /// <param name="yAxisTitle">Title for the Y axis.</param>
00073 /// <param name="title">Title for the plot.</param>
00074 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00075 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00076 /// <param name="boxPresentationAttributes">Presentation attributes for the boxes.</param>
00077 /// <param name="outlierPointElement">The symbol drawn at outlier points.</param>
00078 /// <param name="outlierPresentationAttributes">Presentation attributes for the outlier
      points.</param>
00079 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00080 /// <returns>A <see cref="Plot"/> containing the box plot.</returns>
00081             public static Plot BoxPlot<T>(IReadOnlyList<(T, IReadOnlyList<double>)> data, WhiskerType
      whiskerType = WhiskerType.IQR_1_5, bool useNotches = true, bool proportionalWidth = false, bool
      showOutliers = true, double boxWidth = 10, double spacing = 0.1, double?  dataRangeMin = null, double?
      dataRangeMax = null, bool vertical = true, double width = 350, double height = 250,
00082                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00083                 double axisArrowSize = 3,
00084                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00085                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00086                 string xAxisTitle = null,
00087                 string yAxisTitle = null,
00088                 string title = null,
00089                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00090                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00091                 IReadOnlyList<PlotElementPresentationAttributes> boxPresentationAttributes = null,
00092                 IDataPointElement outlierPointElement = null,
00093                 IReadOnlyList<PlotElementPresentationAttributes> outlierPresentationAttributes = null,
00094                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00095             {
00096                 if (boxPresentationAttributes == null)
00097                 {
00098                     boxPresentationAttributes = new PlotElementPresentationAttributes[] { new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(197, 235, 255)),
      Stroke = new SolidColourBrush(Colour.FromRgb(0, 114, 178)) },
00099                                                                                       new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 233, 218)),
      Stroke = new SolidColourBrush(Colour.FromRgb(213, 94, 0)) },
00100                                                                                       new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 222, 240)),
      Stroke = new SolidColourBrush(Colour.FromRgb(204, 121, 167)) },
00101                                                                                       new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 242, 216)),
```

```
      Stroke = new SolidColourBrush(Colour.FromRgb(230, 159, 0)) },
00102                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(214, 241, 255)),
      Stroke = new SolidColourBrush(Colour.FromRgb(86, 180, 233)) },
00103                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(203, 255, 239)),
      Stroke = new SolidColourBrush(Colour.FromRgb(0, 158, 115)) },
00104                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 249, 189)),
      Stroke = new SolidColourBrush(Colour.FromRgb(240, 228, 66)) } };
00105                 }
00106
00107                 if (outlierPresentationAttributes == null)
00108                 {
00109                     PlotElementPresentationAttributes[] temp = new
      PlotElementPresentationAttributes[boxPresentationAttributes.Count];
00110
00111                     for (int i = 0; i < boxPresentationAttributes.Count; i++)
00112                     {
00113                         temp[i] = new PlotElementPresentationAttributes() { Stroke = null, Fill =
      boxPresentationAttributes[i].Stroke };
00114                     }
00115
00116                     outlierPresentationAttributes = temp;
00117                 }
00118
00119                 if (axisPresentationAttributes == null)
00120                 {
00121                     axisPresentationAttributes = new PlotElementPresentationAttributes();
00122                 }
00123
00124                 if (gridPresentationAttributes == null)
00125                 {
00126                     gridPresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      new SolidColourBrush(Colour.FromRgb(220, 220, 220)) };
00127                 }
00128
00129                 if (axisLabelPresentationAttributes == null)
00130                 {
00131                     axisLabelPresentationAttributes = new PlotElementPresentationAttributes() { Stroke
      = null };
00132                 }
00133
00134                 if (axisTitlePresentationAttributes == null)
00135                 {
00136                     axisTitlePresentationAttributes = new PlotElementPresentationAttributes() { Font =
      new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 14), Stroke =
      null };
00137                 }
00138
00139                 if (titlePresentationAttributes == null)
00140                 {
00141                     titlePresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 18)
      };
00142                 }
00143
00144                 if (outlierPointElement == null)
00145                 {
00146                     outlierPointElement = new PathDataPointElement();
00147                 }
00148
00149                 double[] medians = new double[data.Count];
00150                 double[] firstQuartiles = new double[data.Count];
00151                 double[] thirdQuartiles = new double[data.Count];
00152                 double[] upperWhiskers = new double[data.Count];
00153                 double[] lowerWhiskers = new double[data.Count];
00154                 double[] notches = new double[data.Count];
00155
00156                 double minData = double.MaxValue;
00157                 double maxData = double.MinValue;
00158
00159                 for (int i = 0; i < data.Count; i++)
00160                 {
00161                     medians[i] = data[i].Item2.Median();
00162                     firstQuartiles[i] = data[i].Item2.LowerQuartile();
00163                     thirdQuartiles[i] = data[i].Item2.UpperQuartile();
00164
00165                     if (useNotches)
00166                     {
00167                         notches[i] = 1.58 * (thirdQuartiles[i] - firstQuartiles[i]) /
      Math.Sqrt(data[i].Item2.Count);
00168                     }
00169                     else
00170                     {
00171                         notches[i] = 0;
00172                     }
```

```
00173
00174                     if (whiskerType == WhiskerType.FullRange)
00175                     {
00176                         double uW = double.MinValue;
00177                         double lW = double.MaxValue;
00178
00179                         for (int j = 0; j < data[i].Item2.Count; j++)
00180                         {
00181                             uW = Math.Max(uW, data[i].Item2[j]);
00182                             lW = Math.Min(lW, data[i].Item2[j]);
00183
00184                             minData = Math.Min(minData, data[i].Item2[j]);
00185                             maxData = Math.Max(maxData, data[i].Item2[j]);
00186                         }
00187
00188                         upperWhiskers[i] = uW;
00189                         lowerWhiskers[i] = lW;
00190                     }
00191                     else if (whiskerType == WhiskerType.IQR_1_5)
00192                     {
00193                         upperWhiskers[i] = thirdQuartiles[i] + (thirdQuartiles[i] - firstQuartiles[i])
    * 1.5;
00194                         lowerWhiskers[i] = firstQuartiles[i] - (thirdQuartiles[i] - firstQuartiles[i])
    * 1.5;
00195
00196                         double uW = double.MinValue;
00197                         double lW = double.MaxValue;
00198
00199                         for (int j = 0; j < data[i].Item2.Count; j++)
00200                         {
00201                             if (data[i].Item2[j] <= upperWhiskers[i])
00202                             {
00203                                 uW = Math.Max(uW, data[i].Item2[j]);
00204                             }
00205
00206                             if (data[i].Item2[j] >= lowerWhiskers[i])
00207                             {
00208                                 lW = Math.Min(lW, data[i].Item2[j]);
00209                             }
00210
00211                             minData = Math.Min(minData, data[i].Item2[j]);
00212                             maxData = Math.Max(maxData, data[i].Item2[j]);
00213                         }
00214
00215                         upperWhiskers[i] = uW;
00216                         lowerWhiskers[i] = lW;
00217                     }
00218                     else if (whiskerType == WhiskerType.StandardDeviation)
00219                     {
00220                         double stdDev = data[i].Item2.StandardDeviation();
00221                         double mean = data[i].Item2.Mean();
00222
00223                         upperWhiskers[i] = mean + 2 * stdDev;
00224                         lowerWhiskers[i] = mean - 2 * stdDev;
00225
00226                         double uW = double.MinValue;
00227                         double lW = double.MaxValue;
00228
00229                         for (int j = 0; j < data[i].Item2.Count; j++)
00230                         {
00231                             if (data[i].Item2[j] <= upperWhiskers[i])
00232                             {
00233                                 uW = Math.Max(uW, data[i].Item2[j]);
00234                             }
00235
00236                             if (data[i].Item2[j] >= lowerWhiskers[i])
00237                             {
00238                                 lW = Math.Min(lW, data[i].Item2[j]);
00239                             }
00240
00241                             minData = Math.Min(minData, data[i].Item2[j]);
00242                             maxData = Math.Max(maxData, data[i].Item2[j]);
00243                         }
00244
00245                         upperWhiskers[i] = uW;
00246                         lowerWhiskers[i] = lW;
00247                     }
00248                 }
00249
00250                 if (!showOutliers)
00251                 {
00252                     minData = lowerWhiskers.Minimum();
00253                     maxData = upperWhiskers.Maximum();
00254                 }
00255
00256                 double minX, maxX, minY, maxY;
00257
```

```
00258                 if (vertical)
00259                 {
00260                     minX = 0;
00261                     maxX = boxWidth * (data.Count + spacing * (data.Count - 1));
00262 
00263                     minY = minData;
00264                     maxY = maxData;
00265 
00266                     if (dataRangeMin != null)
00267                     {
00268                         minY = dataRangeMin.Value;
00269                     }
00270 
00271                     if (dataRangeMax != null)
00272                     {
00273                         maxY = dataRangeMax.Value;
00274                     }
00275                 }
00276                 else
00277                 {
00278                     minY = 0;
00279                     maxY = boxWidth * (data.Count + spacing * (data.Count - 1));
00280 
00281                     minX = minData;
00282                     maxX = maxData;
00283 
00284                     if (dataRangeMin != null)
00285                     {
00286                         minX = dataRangeMin.Value;
00287                     }
00288 
00289                     if (dataRangeMax != null)
00290                     {
00291                         maxX = dataRangeMax.Value;
00292                     }
00293                 }
00294 
00295 
00296                 if (coordinateSystem == null)
00297                 {
00298                     coordinateSystem = new LinearCoordinateSystem2D(minX, maxX, minY, maxY, width,
      height);
00299                 }
00300 
00301                 Point topLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, maxY });
00302                 Point topRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, maxY });
00303                 Point bottomRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, minY });
00304                 Point bottomLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, minY });
00305 
00306                 double[] marginTopLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00307                 double[] marginTopRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00308                 double[] marginBottomRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00309                 double[] marginBottomLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00310 
00311                 double[] p1 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00312                 double[] p2 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00313                 double[] p3 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00314                 double[] p4 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00315 
00316 
00317                 double[] p5 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00318                 double[] p6 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00319                 double[] p7 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00320                 double[] p8 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
```

```
    Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00321
00322                 ContinuousAxis xAxis = new ContinuousAxis(marginBottomLeft[0] < marginBottomRight[0] ?
    marginBottomLeft :  marginBottomRight, marginBottomLeft[0] < marginBottomRight[0] ?  marginBottomRight
    :  vertical ?  marginBottomLeft :
    coordinateSystem.ToDataCoordinates(coordinateSystem.ToPlotCoordinates(marginBottomLeft) + new
    Point(-axisArrowSize - 7, 0)), coordinateSystem) { PresentationAttributes =
    axisPresentationAttributes, ArrowSize = vertical ?  0 :  axisArrowSize };
00323                 ContinuousAxis yAxis = new ContinuousAxis(marginBottomLeft[1] < marginTopLeft[1] ?
    marginBottomLeft :  marginTopLeft, marginBottomLeft[1] < marginTopLeft[1] ?  marginTopLeft :
    !vertical ?  marginBottomLeft :
    coordinateSystem.ToDataCoordinates(coordinateSystem.ToPlotCoordinates(marginBottomLeft) + new Point(0,
    axisArrowSize + 7)), coordinateSystem) { PresentationAttributes = axisPresentationAttributes,
    ArrowSize = vertical ?  axisArrowSize :  0 };
00324
00325                 IPlotElement xTicks;
00326                 IPlotElement yTicks;
00327
00328                 IPlotElement xLabels;
00329                 IPlotElement yLabels;
00330
00331                 if (vertical)
00332                 {
00333                     //xTicks = new ContinuousAxisTicks(new double[] { 0.5 * boxWidth, p3[1] }, new
    double[] { ((data.Count - 1) * (spacing + 1) + 0.5) * boxWidth, p4[1] }, coordinateSystem) {
    PresentationAttributes = axisPresentationAttributes, IntervalCount = data.Count - 1, SizeAbove = _ =>
    3, SizeBelow = _ => 3 };
00334
00335                     xTicks = new ScatterPoints<IReadOnlyList<double>>((from el in Enumerable.Range(0,
    data.Count) select new double[] { (el * (spacing + 1) + 0.5) * boxWidth, p3[1] }), coordinateSystem) {
    PresentationAttributes = axisPresentationAttributes, Size = 1, DataPointElement = new
    PathDataPointElement(new GraphicsPath().MoveTo(0, -3).LineTo(0, 3)) };
00336                     yTicks = new ContinuousAxisTicks(p6, p5, coordinateSystem) {
    PresentationAttributes = axisPresentationAttributes };
00337
00338                     xLabels = new DataLabels<IReadOnlyList<double>>((from el in Enumerable.Range(0,
    data.Count) select new double[] { (el * (spacing + 1) + 0.5) * boxWidth, p3[1] }), coordinateSystem) {
    Alignment = TextAnchors.Center, Baseline = TextBaselines.Top, Label = (i, _) => data[i].Item1,
    PresentationAttributes = axisLabelPresentationAttributes, Margin = (a, b) => new Point(0, 10) };
00339                     yLabels = new ContinuousAxisLabels(p6, p5, coordinateSystem) {
    PresentationAttributes = axisLabelPresentationAttributes, Position = _ => -10, Alignment =
    TextAnchors.Right, Rotation = 0, IntervalCount = 5 };
00340                 }
00341                 else
00342                 {
00343                     xTicks = new ContinuousAxisTicks(p3, p4, coordinateSystem) {
    PresentationAttributes = axisPresentationAttributes };
00344
00345                     yTicks = new ScatterPoints<IReadOnlyList<double>>((from el in Enumerable.Range(0,
    data.Count) select new double[] { p6[0], (el * (spacing + 1) + 0.5) * boxWidth }), coordinateSystem) {
    PresentationAttributes = axisPresentationAttributes, Size = 1, DataPointElement = new
    PathDataPointElement(new GraphicsPath().MoveTo(-3, 0).LineTo(3, 0)) };
00346
00347                     xLabels = new ContinuousAxisLabels(p3, p4, coordinateSystem) {
    PresentationAttributes = axisLabelPresentationAttributes, Alignment = TextAnchors.Center, Baseline =
    TextBaselines.Top, Rotation = 0, IntervalCount = 5 };
00348                     yLabels = new DataLabels<IReadOnlyList<double>>((from el in Enumerable.Range(0,
    data.Count) select new double[] { p6[0], (el * (spacing + 1) + 0.5) * boxWidth }), coordinateSystem) {
    Alignment = TextAnchors.Right, Baseline = TextBaselines.Middle, Label = (i, _) => data[i].Item1,
    PresentationAttributes = axisLabelPresentationAttributes, Margin = (a, b) => new Point(-10, 0) };
00349                 }
00350
00351                 Graphics xLabelsSize = new Graphics();
00352                 xLabels.Plot(xLabelsSize);
00353                 double xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
00354
00355                 Graphics yLabelsSize = new Graphics();
00356                 yLabels.Plot(yLabelsSize);
00357                 double yLabelsWidth = yLabelsSize.GetBounds().Size.Width;
00358
00359                 ContinuousAxisTitle xTitle = new ContinuousAxisTitle(xAxisTitle, marginBottomLeft,
    marginBottomRight, coordinateSystem, axisTitlePresentationAttributes) { Position = xLabelsHeight + 20,
    Baseline = TextBaselines.Top, Alignment = TextAnchors.Center };
00360                 ContinuousAxisTitle yTitle = new ContinuousAxisTitle(yAxisTitle, marginBottomLeft,
    marginTopLeft, coordinateSystem, axisTitlePresentationAttributes) { Position = -20 - yLabelsWidth,
    Baseline = TextBaselines.Bottom, Alignment = TextAnchors.Center };
00361
00362                 Plot plot = new Plot();
00363
00364                 if (vertical)
00365                 {
00366                     Grid yGrid = new Grid(p5, p6, p7, p8, coordinateSystem) { IntervalCount = 5,
    PresentationAttributes = gridPresentationAttributes };
00367                     plot.AddPlotElement(yGrid);
00368                 }
00369                 else
00370                 {
```

```
00371                     Grid xGrid = new Grid(p1, p2, p3, p4, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00372                     plot.AddPlotElement(xGrid);
00373                 }
00374 
00375                 plot.AddPlotElements(xAxis, yAxis, xTicks, yTicks, xLabels, yLabels, xTitle, yTitle);
00376 
00377                 double maxCount = (from el in data select el.Item2.Count).Max();
00378 
00379                 for (int i = 0; i < data.Count; i++)
00380                 {
00381                     if (vertical)
00382                     {
00383                         BoxPlot box = new BoxPlot(new double[] { (i * (spacing + 1) + 0.5) * boxWidth,
      medians[i] }, new double[] { 0, 1 }, lowerWhiskers[i] - medians[i], firstQuartiles[i] - medians[i],
      thirdQuartiles[i] - medians[i], upperWhiskers[i] - medians[i], coordinateSystem)
00384                         {
00385                             BoxPresentationAttributes = boxPresentationAttributes[i %
      boxPresentationAttributes.Count],
00386                             WhiskersPresentationAttributes = boxPresentationAttributes[i %
      boxPresentationAttributes.Count],
00387                             Width = proportionalWidth ?  (boxWidth * 0.5 * data[i].Item2.Count /
      maxCount) :  (boxWidth * 0.5),
00388                             NotchSize = notches[i]
00389                         };
00390 
00391                         plot.AddPlotElement(box);
00392                     }
00393                     else
00394                     {
00395                         BoxPlot box = new BoxPlot(new double[] { medians[i], (i * (spacing + 1) + 0.5)
      * boxWidth }, new double[] { 1, 0 }, lowerWhiskers[i] - medians[i], firstQuartiles[i] - medians[i],
      thirdQuartiles[i] - medians[i], upperWhiskers[i] - medians[i], coordinateSystem)
00396                         {
00397                             BoxPresentationAttributes = boxPresentationAttributes[i %
      boxPresentationAttributes.Count],
00398                             WhiskersPresentationAttributes = boxPresentationAttributes[i %
      boxPresentationAttributes.Count],
00399                             Width = proportionalWidth ?  (boxWidth * 0.5 * data[i].Item2.Count /
      maxCount) :  (boxWidth * 0.5),
00400                             NotchSize = notches[i]
00401                         };
00402 
00403                         plot.AddPlotElement(box);
00404                     }
00405                 }
00406 
00407                 if (showOutliers)
00408                 {
00409                     for (int i = 0; i < data.Count; i++)
00410                     {
00411                         double[][] outliers;
00412 
00413                         if (vertical)
00414                         {
00415                             outliers = (from el in data[i].Item2 where el > upperWhiskers[i] || el <
      lowerWhiskers[i] select new double[] { (i * (spacing + 1) + 0.5) * boxWidth, el }).ToArray();
00416                         }
00417                         else
00418                         {
00419                             outliers = (from el in data[i].Item2 where el > upperWhiskers[i] || el <
      lowerWhiskers[i] select new double[] { el, (i * (spacing + 1) + 0.5) * boxWidth }).ToArray();
00420                         }
00421 
00422                         if (outliers.Length > 0)
00423                         {
00424                             ScatterPoints<IReadOnlyList<double» outlierPoints = new
      ScatterPoints<IReadOnlyList<double»(outliers, coordinateSystem)
00425                             {
00426                                 PresentationAttributes = outlierPresentationAttributes[i %
      outlierPresentationAttributes.Count],
00427                                 Size = 3,
00428                                 DataPointElement= outlierPointElement
00429                             };
00430 
00431                             plot.AddPlotElement(outlierPoints);
00432                         }
00433                     }
00434                 }
00435 
00436                 TextLabel<IReadOnlyList<double» titleLabel = new
      TextLabel<IReadOnlyList<double»(title,
      coordinateSystem.ToDataCoordinates((coordinateSystem.ToPlotCoordinates(marginTopLeft) +
      coordinateSystem.ToPlotCoordinates(marginTopRight)) * 0.5 + new Point(0, -10)), coordinateSystem) {
      Baseline = TextBaselines.Bottom, PresentationAttributes = titlePresentationAttributes };
00437 
00438                 plot.AddPlotElement(titleLabel);
```

```
00439
00440                 return plot;
00441             }
00442
00443 /// <summary>
00444 /// Create a new box plot.
00445 /// </summary>
00446 /// <param name="data">The data to plot.</param>
00447 /// <param name="vertical">If this is <see langword="true"/> (the default), the boxes are parallel to
      the Y axis.  If this is <see langword="false"/>, the boxes are parallel to the X axis.</param>
00448 /// <param name="whiskerType">The type of whiskers to use in the plot.</param>
00449 /// <param name="useNotches">If this is <see langword="true"/> (the default), the box plot has
      notches.  Otherwise, it does not.</param>
00450 /// <param name="proportionalWidth">If this is <see langword="false"/> (the default), all the boxes
      have the same width.  Otherwise, the width of each box is proportional to the number of
      samples.</param>
00451 /// <param name="showOutliers">If this is <see langword="true"/> (the default), a symbol is drawn four
      outlier points that fall outside of the interval between the whiskers.</param>
00452 /// <param name="boxWidth">The width of the boxes in data space coordinates.</param>
00453 /// <param name="spacing">The spacing between consecutive boxes (ranging from 0 to 1).</param>
00454 /// <param name="dataRangeMin">If this is not <see langword="null"/>, this value is used to override
      the default minimum value for the plot.  Useful if you wish to create multiple plots with the same
      scale,
00455 /// even though the sampled values have different ranges.</param>
00456 /// <param name="dataRangeMax">If this is not <see langword="null"/>, this value is used to override
      the default maximum value for the plot.  Useful if you wish to create multiple plots with the same
      scale,
00457 /// even though the sampled values have different ranges.</param>
00458 /// <param name="width">The width of the plot.</param>
00459 /// <param name="height">The height of the plot.</param>
00460 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00461 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00462 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00463 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00464 /// <param name="xAxisTitle">Title for the X axis.</param>
00465 /// <param name="yAxisTitle">Title for the Y axis.</param>
00466 /// <param name="title">Title for the plot.</param>
00467 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00468 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00469 /// <param name="boxPresentationAttributes">Presentation attributes for the boxes.</param>
00470 /// <param name="outlierPointElement">The symbol drawn at outlier points.</param>
00471 /// <param name="outlierPresentationAttributes">Presentation attributes for the outlier
      points.</param>
00472 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00473 /// <returns>A <see cref="Plot"/> containing the box plot.</returns>
00474             public static Plot BoxPlot(IReadOnlyList<IReadOnlyList<double» data, WhiskerType
      whiskerType = WhiskerType.IQR_1_5, bool useNotches = true, bool proportionalWidth = false, bool
      showOutliers = true, double boxWidth = 10, double spacing = 0.1, double?  dataRangeMin = null, double?
      dataRangeMax = null, bool vertical = true, double width = 350, double height = 250,
00475                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00476                 double axisArrowSize = 3,
00477                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00478                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00479                 string xAxisTitle = null,
00480                 string yAxisTitle = null,
00481                 string title = null,
00482                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00483                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00484                 IReadOnlyList<PlotElementPresentationAttributes> boxPresentationAttributes = null,
00485                 IDataPointElement outlierPointElement = null,
00486                 IReadOnlyList<PlotElementPresentationAttributes> outlierPresentationAttributes = null,
00487                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00488             {
00489                 return BoxPlot((from el in data select ((string)null, el)).ToArray(), whiskerType,
      useNotches, proportionalWidth, showOutliers, boxWidth, spacing, dataRangeMin, dataRangeMax, vertical,
      width, height, axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, boxPresentationAttributes, outlierPointElement,
      outlierPresentationAttributes, coordinateSystem);
00490             }
00491
00492 /// <summary>
00493 /// Create a new box plot.
00494 /// </summary>
00495 /// <param name="data">The data to plot.</param>
00496 /// <param name="vertical">If this is <see langword="true"/> (the default), the box is parallel to the
      Y axis.  If this is <see langword="false"/>, the box is parallel to the X axis.</param>
00497 /// <param name="whiskerType">The type of whiskers to use in the plot.</param>
00498 /// <param name="useNotches">If this is <see langword="true"/> (the default), the box plot has
      notches.  Otherwise, it does not.</param>
00499 /// <param name="showOutliers">If this is <see langword="true"/> (the default), a symbol is drawn four
      outlier points that fall outside of the interval between the whiskers.</param>
00500 /// <param name="boxWidth">The width of the box in data space coordinates.</param>
00501 /// <param name="dataRangeMin">If this is not <see langword="null"/>, this value is used to override
      the default minimum value for the plot.  Useful if you wish to create multiple plots with the same
      scale,
```

```
00502 /// even though the sampled values have different ranges.</param>
00503 /// <param name="dataRangeMax">If this is not <see langword="null"/>, this value is used to override
      the default maximum value for the plot.  Useful if you wish to create multiple plots with the same
      scale,
00504 /// even though the sampled values have different ranges.</param>
00505 /// <param name="width">The width of the plot.</param>
00506 /// <param name="height">The height of the plot.</param>
00507 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00508 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00509 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00510 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00511 /// <param name="xAxisTitle">Title for the X axis.</param>
00512 /// <param name="yAxisTitle">Title for the Y axis.</param>
00513 /// <param name="title">Title for the plot.</param>
00514 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00515 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00516 /// <param name="boxPresentationAttributes">Presentation attributes for the box.</param>
00517 /// <param name="outlierPointElement">The symbol drawn at outlier points.</param>
00518 /// <param name="outlierPresentationAttributes">Presentation attributes for the outlier
      points.</param>
00519 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00520 /// <returns>A <see cref="Plot"/> containing the box plot.</returns>
00521             public static Plot BoxPlot(IReadOnlyList<double> data, WhiskerType whiskerType =
      WhiskerType.IQR_1_5, bool useNotches = true, bool showOutliers = true, double boxWidth = 10, double?
      dataRangeMin = null, double?  dataRangeMax = null, bool vertical = true, double width = 350, double
      height = 250,
00522                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00523                 double axisArrowSize = 3,
00524                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00525                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00526                 string xAxisTitle = null,
00527                 string yAxisTitle = null,
00528                 string title = null,
00529                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00530                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00531                 PlotElementPresentationAttributes boxPresentationAttributes = null,
00532                 IDataPointElement outlierPointElement = null,
00533                 PlotElementPresentationAttributes outlierPresentationAttributes = null,
00534                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00535             {
00536                 return BoxPlot(new (string, IReadOnlyList<double>)[] { (null, data) }, whiskerType,
      useNotches, false, showOutliers, boxWidth, 0, dataRangeMin, dataRangeMax, vertical, width, height,
      axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, boxPresentationAttributes == null ?  null :  new
      PlotElementPresentationAttributes[] { boxPresentationAttributes }, outlierPointElement,
      outlierPresentationAttributes == null ?  null :  new PlotElementPresentationAttributes[] {
      outlierPresentationAttributes }, coordinateSystem);
00537             }
00538         }
00539     }
00540 }
```

8.46 Plot.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Collections.Immutable;
00021
00022 namespace VectSharp.Plots
00023 {
00024 /// <summary>
00025 /// Determines the appearance of plot elements.
00026 /// </summary>
00027     public class PlotElementPresentationAttributes
```

```
00028     {
00029 /// <summary>
00030 /// The stroke of the plot element.
00031 /// </summary>
00032         public Brush Stroke { get; set; } = Colours.Black;
00033 
00034 /// <summary>
00035 /// The fill of the plot element.
00036 /// </summary>
00037         public Brush Fill { get; set; } = Colours.Black;
00038 
00039 /// <summary>
00040 /// The thickness of lines in the plot element.
00041 /// </summary>
00042         public double LineWidth { get; set; } = 1;
00043 
00044 /// <summary>
00045 /// The line dash style for the plot element.
00046 /// </summary>
00047         public LineDash?  LineDash { get; set; } = null;
00048 
00049 /// <summary>
00050 /// The line cap for the plot element.
00051 /// </summary>
00052         public LineCaps LineCap { get; set; } = LineCaps.Square;
00053 
00054 /// <summary>
00055 /// The line join for the plot element.
00056 /// </summary>
00057         public LineJoins LineJoin { get; set; } = LineJoins.Miter;
00058 
00059 /// <summary>
00060 /// The font used to draw text in the plot element.
00061 /// </summary>
00062         public Font Font { get; set; } = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica), 12);
00063 
00064 /// <summary>
00065 /// Create a new <see cref="PlotElementPresentationAttributes"/> with the default values.
00066 /// </summary>
00067         public PlotElementPresentationAttributes()
00068         {
00069 
00070         }
00071 
00072 /// <summary>
00073 /// Creates a new <see cref="PlotElementPresentationAttributes"/> copying all settings from another
      instance.
00074 /// </summary>
00075 /// <param name="other">The other instance from which the settings will be copied.</param>
00076         public PlotElementPresentationAttributes(PlotElementPresentationAttributes other)
00077         {
00078             this.LineJoin = other.LineJoin;
00079             this.Stroke = other.Stroke;
00080             this.Fill = other.Fill;
00081             this.LineCap = other.LineCap;
00082             this.LineDash = other.LineDash;
00083             this.Font = other.Font;
00084             this.LineWidth = other.LineWidth;
00085         }
00086     }
00087 
00088 /// <summary>
00089 /// Represents a plot element.
00090 /// </summary>
00091     public interface IPlotElement
00092     {
00093 /// <summary>
00094 /// Draw the plot element on the specified <paramref name="target"/> <see cref="Graphics"/>.
00095 /// </summary>
00096 /// <param name="target">The <see cref="Graphics"/> on which to draw.</param>
00097         void Plot(Graphics target);
00098 
00099 /// <summary>
00100 /// The coordinate system used to transform the points from data space to plot space.
00101 /// </summary>
00102         ICoordinateSystem CoordinateSystem { get; }
00103     }
00104 
00105 /// <summary>
00106 /// A plot element that uses an <see cref="Action"/> to draw its contents.
00107 /// </summary>
00108 /// <typeparam name="T">The type of data to plot (e.g.  <c>double[]</c>).</typeparam>
00109     public class PlotElement<T> :  IPlotElement
00110     {
00111 /// <summary>
00112 /// The action that is invoked when the plot element needs to be drawn.
```

```
00113 /// </summary>
00114         public Action<Graphics, ICoordinateSystem<T» PlotAction { get; set; }
00115
00116 /// <summary>
00117 /// The coordinate system used to transform the points from data space to plot space.
00118 /// </summary>
00119         public ICoordinateSystem<T> CoordinateSystem { get; set; }
00120         ICoordinateSystem IPlotElement.CoordinateSystem => this.CoordinateSystem;
00121
00122 /// <summary>
00123 /// Create a new <see cref="PlotElement{T}"/> using the specified coordinate system and plot action.
00124 /// </summary>
00125 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00126 /// <param name="plotAction">The action that is invoked when the plot element needs to be
      drawn.</param>
00127         public PlotElement(ICoordinateSystem<T> coordinateSystem, Action<Graphics,
      ICoordinateSystem<T>> plotAction)
00128         {
00129             PlotAction = plotAction;
00130             CoordinateSystem = coordinateSystem;
00131         }
00132
00133 /// <inheritdoc/>
00134         public void Plot(Graphics target)
00135         {
00136             PlotAction?.Invoke(target, CoordinateSystem);
00137         }
00138     }
00139
00140 /// <summary>
00141 /// Represents a collection of plot elements.
00142 /// </summary>
00143     public partial class Plot
00144     {
00145 /// <summary>
00146 /// The elements contained in the plot.
00147 /// </summary>
00148         public ImmutableList<IPlotElement> PlotElements { get; private set; } =
      ImmutableList.Create<IPlotElement>();
00149
00150 /// <summary>
00151 /// Add the specified plot element to the plot.
00152 /// </summary>
00153 /// <param name="plotElement">The plot element to add.</param>
00154         public void AddPlotElement(IPlotElement plotElement)
00155         {
00156             this.PlotElements = this.PlotElements.Add(plotElement);
00157         }
00158
00159 /// <summary>
00160 /// Add the specified plot elements to the plot.
00161 /// </summary>
00162 /// <param name="plotElements">The plot elements to add.</param>
00163         public void AddPlotElements(IEnumerable<IPlotElement> plotElements)
00164         {
00165             this.PlotElements = this.PlotElements.AddRange(plotElements);
00166         }
00167
00168 /// <summary>
00169 /// Add the specified plot elements to the plot.
00170 /// </summary>
00171 /// <param name="plotElements">The plot elements to add.</param>
00172         public void AddPlotElements(params IPlotElement[] plotElements)
00173         {
00174             this.AddPlotElements((IEnumerable<IPlotElement>)plotElements);
00175         }
00176
00177 /// <summary>
00178 /// Remove the specified plot element from the plot.
00179 /// </summary>
00180 /// <param name="plotElement">The plot element to remove.</param>
00181         public void RemovePlotElement(IPlotElement plotElement)
00182         {
00183             this.PlotElements = this.PlotElements.Remove(plotElement);
00184         }
00185
00186 /// <summary>
00187 /// Render the plot on the specified <paramref name="target"/> <see cref="Graphics"/>.
00188 /// </summary>
00189 /// <param name="target">The <see cref="Graphics"/> on which the plot should be drawn.</param>
00190         public void Render(Graphics target)
00191         {
00192             foreach (IPlotElement element in PlotElements)
00193             {
00194                 element.Plot(target);
00195             }
```

```
00196         }
00197
00198 /// <summary>
00199 /// Get the first <see cref="IPlotElement"/> of the specified type, or the first <see
      cref="ICoordinateSystem"/> of the specified type.
00200 /// </summary>
00201 /// <typeparam name="T">The type of plot element or coordinate system to get.</typeparam>
00202 /// <returns>The first <see cref="IPlotElement"/> of the specified type, or the first <see
      cref="ICoordinateSystem"/> of the specified type, or <see langword="null"/> if none could be
      found.</returns>
00203         public T GetFirst<T>() where T : class
00204         {
00205             foreach (IPlotElement e in this.PlotElements)
00206             {
00207                 if (e is T t)
00208                 {
00209                     return t;
00210                 }
00211                 else if (e.CoordinateSystem is T t2)
00212                 {
00213                     return t2;
00214                 }
00215             }
00216
00217             return null;
00218         }
00219
00220 /// <summary>
00221 /// Get all <see cref="IPlotElement"/>s of the specified type, or all <see cref="ICoordinateSystem"/>s
      of the specified type.
00222 /// </summary>
00223 /// <typeparam name="T">The type of plot element or coordinate system to get.</typeparam>
00224 /// <returns>All <see cref="IPlotElement"/>s of the specified type, or all <see
      cref="ICoordinateSystem"/>s of the specified type.</returns>
00225         public IEnumerable<T> GetAll<T>() where T : class, IPlotElement
00226         {
00227             foreach (IPlotElement e in this.PlotElements)
00228             {
00229                 if (e is T t)
00230                 {
00231                     yield return t;
00232                 }
00233                 else if (e.CoordinateSystem is T t2)
00234                 {
00235                     yield return t2;
00236                 }
00237             }
00238         }
00239
00240 /// <summary>
00241 /// Render the plot to a suitably cropped <see cref="Page"/> object.
00242 /// </summary>
00243 /// <returns>A <see cref="Page"/> object containing the rendered plot.</returns>
00244         public Page Render()
00245         {
00246             Page pag = new Page(1, 1);
00247
00248             this.Render(pag.Graphics);
00249
00250             Rectangle bounds = pag.Graphics.GetBounds();
00251
00252             pag.Crop(new Point(bounds.Location.X - bounds.Size.Width * 0.01, bounds.Location.Y -
      bounds.Size.Height * 0.01), new Size(bounds.Size.Width * 1.02, bounds.Size.Height * 1.02));
00253
00254             return pag;
00255         }
00256
00257 /// <summary>
00258 /// Create a new empty <see cref="Plot"/>.
00259 /// </summary>
00260         public Plot()
00261         {
00262
00263         }
00264
00265         private static (double minX, double minY, double maxX, double maxY, double rangeX, double
      rangeY) GetDataRange(IReadOnlyList<IReadOnlyList<double>> data)
00266         {
00267             double minX = double.MaxValue;
00268             double minY = double.MaxValue;
00269
00270             double maxX = double.MinValue;
00271             double maxY = double.MinValue;
00272
00273             for (int i = 0; i < data.Count; i++)
00274             {
00275                 minX = Math.Min(minX, data[i][0]);
```

```
00276                 minY = Math.Min(minY, data[i][1]);
00277
00278                 maxX = Math.Max(maxX, data[i][0]);
00279                 maxY = Math.Max(maxY, data[i][1]);
00280             }
00281
00282             double rangeX = maxX - minX;
00283             double rangeY = maxY - minY;
00284
00285             return (minX, minY, maxX, maxY, rangeX, rangeY);
00286         }
00287
00288 /// <summary>
00289 /// Contains methods to create plots.
00290 /// </summary>
00291         public static partial class Create
00292         {
00293 /// <summary>
00294 /// Creates a new empty plot.
00295 /// </summary>
00296 /// <returns>An empty plot.</returns>
00297             public static Plot Empty()
00298             {
00299                 return new Plot();
00300             }
00301         }
00302     }
00303 }
```

8.47 Plot.Function1D.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020
00021 namespace VectSharp.Plots
00022 {
00023     partial class Plot
00024     {
00025         partial class Create
00026         {
00027 /// <summary>
00028 /// Create a new plot for functions of one variable.
00029 /// </summary>
00030 /// <param name="functions">The functions to plot.</param>
00031 /// <param name="min">The minimum value of the function argument to plot.</param>
00032 /// <param name="max">The maximum value of the function argument to plot.</param>
00033 /// <param name="resolution">The distance between consecutive function samples.</param>
00034 /// <param name="vertical">If this is <see langword="true"/> (the default), the functions are plotted
      as y = f(x).  If this is <see langword="false"/>, the functions are plotted as x = f(y).</param>
00035 /// <param name="smooth">If this is <see langword="false"/>, consecutive function samples are joined
      by a polyline.  If this is <see langword="true"/>, they are joined by a smooth spline passing through
      them.</param>
00036 /// <param name="width">The width of the plot.</param>
00037 /// <param name="height">The height of the plot.</param>
00038 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00039 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00040 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00041 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00042 /// <param name="xAxisTitle">Title for the X axis.</param>
00043 /// <param name="yAxisTitle">Title for the Y axis.</param>
00044 /// <param name="title">Title for the plot.</param>
00045 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00046 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00047 /// <param name="linePresentationAttributes">Presentation attributes for the function line.</param>
00048 /// <param name="pointPresentationAttributes">Presentation attributes for the symbols drawn at the
      sampled function points.</param>
```

```
00049 /// <param name="pointSize">Size of the symbols drawn at the sampled function points.</param>
00050 /// <param name="dataPointElements">Symbols drawn at the sampled function points.</param>
00051 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00052 /// <returns>A <see cref="Plot"/> containing the bar chart.</returns>
00053             public static Plot Function(IReadOnlyList<Func<double, double» functions, double min,
      double max, double resolution = double.NaN, bool vertical = true, bool smooth = false, double width =
      350, double height = 250,
00054                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00055                 double axisArrowSize = 3,
00056                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00057                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00058                 string xAxisTitle = null,
00059                 string yAxisTitle = null,
00060                 string title = null,
00061                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00062                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00063                 IReadOnlyList<PlotElementPresentationAttributes> linePresentationAttributes = null,
00064                 IReadOnlyList<PlotElementPresentationAttributes> pointPresentationAttributes = null,
00065                 IReadOnlyList<double> pointSize = null,
00066                 IReadOnlyList<IDataPointElement> dataPointElements = null,
00067                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00068             {
00069                 if (double.IsNaN(resolution))
00070                 {
00071                     if (coordinateSystem == null)
00072                     {
00073                         resolution = (max - min) / 500;
00074                     }
00075                     else
00076                     {
00077                         resolution = coordinateSystem.Resolution[0];
00078                     }
00079                 }
00080 
00081                 int steps = (int)Math.Ceiling((max - min) / resolution);
00082 
00083                 List<double[]>[] data = new List<double[]>[functions.Count];
00084 
00085                 for (int i = 0; i < functions.Count; i++)
00086                 {
00087                     data[i] = new List<double[]>(steps + 1);
00088 
00089                     for (int j = 0; j <= steps; j++)
00090                     {
00091                         double x = min + (max - min) / steps * j;
00092                         double y = functions[i](x);
00093 
00094                         if (!double.IsNaN(y))
00095                         {
00096                             if (vertical)
00097                             {
00098                                 data[i].Add(new double[] { x, y });
00099                             }
00100                             else
00101                             {
00102                                 data[i].Add(new double[] { y, x });
00103                             }
00104                         }
00105                     }
00106                 }
00107 
00108                 return Create.LineCharts(data, smooth, width, height, axisPresentationAttributes,
      axisArrowSize, axisLabelPresentationAttributes, axisTitlePresentationAttributes, xAxisTitle,
      yAxisTitle, title, titlePresentationAttributes, gridPresentationAttributes,
      linePresentationAttributes, pointPresentationAttributes, pointSize, dataPointElements,
      coordinateSystem);
00109             }
00110 
00111 /// <summary>
00112 /// Create a new plot for a function of one variable.
00113 /// </summary>
00114 /// <param name="function">The function to plot.</param>
00115 /// <param name="min">The minimum value of the function argument to plot.</param>
00116 /// <param name="max">The maximum value of the function argument to plot.</param>
00117 /// <param name="resolution">The distance between consecutive function samples.</param>
00118 /// <param name="vertical">If this is <see langword="true"/> (the default), the function is plotted as
      y = f(x).  If this is <see langword="false"/>, the function is plotted as x = f(y).</param>
00119 /// <param name="smooth">If this is <see langword="false"/>, consecutive function samples are joined
      by a polyline.  If this is <see langword="true"/>, they are joined by a smooth spline passing through
      them.</param>
00120 /// <param name="width">The width of the plot.</param>
00121 /// <param name="height">The height of the plot.</param>
00122 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00123 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00124 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00125 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
```

```
00126 /// <param name="xAxisTitle">Title for the X axis.</param>
00127 /// <param name="yAxisTitle">Title for the Y axis.</param>
00128 /// <param name="title">Title for the plot.</param>
00129 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00130 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00131 /// <param name="linePresentationAttributes">Presentation attributes for the function line.</param>
00132 /// <param name="pointPresentationAttributes">Presentation attributes for the symbols drawn at the
      sampled function points.</param>
00133 /// <param name="pointSize">Size of the symbols drawn at the sampled function points.</param>
00134 /// <param name="dataPointElement">Symbol drawn at the sampled function points.</param>
00135 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00136 /// <returns>A <see cref="Plot"/> containing the bar chart.</returns>
00137             public static Plot Function(Func<double, double> function, double min, double max, double
      resolution = double.NaN, bool vertical = true, bool smooth = false, double width = 350, double height
      = 250,
00138                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00139                 double axisArrowSize = 3,
00140                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00141                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00142                 string xAxisTitle = null,
00143                 string yAxisTitle = null,
00144                 string title = null,
00145                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00146                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00147                 PlotElementPresentationAttributes linePresentationAttributes = null,
00148                 PlotElementPresentationAttributes pointPresentationAttributes = null,
00149                 double pointSize = 0,
00150                 IDataPointElement dataPointElement = null,
00151                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00152             {
00153                 return Function(new Func<double, double>[] { function }, min, max, resolution,
      vertical, smooth, width, height, axisPresentationAttributes, axisArrowSize,
      axisLabelPresentationAttributes, axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title,
      titlePresentationAttributes, gridPresentationAttributes, linePresentationAttributes == null ?  null :
      new PlotElementPresentationAttributes[] { linePresentationAttributes }, pointPresentationAttributes ==
      null ?  null :  new PlotElementPresentationAttributes[] { pointPresentationAttributes }, new double[]
      { pointSize }, dataPointElement == null ?  null :  new IDataPointElement[] { dataPointElement },
      coordinateSystem);
00154             }
00155         }
00156     }
00157 }
```

8.48 Plot.Function2D.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020
00021 namespace VectSharp.Plots
00022 {
00023     partial class Plot
00024     {
00025         partial class Create
00026         {
00027 /// <summary>
00028 /// Create a new plot for a function of two variables.
00029 /// </summary>
00030 /// <param name="function">The function grid to plot.</param>
00031 /// <param name="plotType">The type of plot to produce.</param>
00032 /// <param name="rasterResolutionX">If <paramref name="plotType"/> is <see
      cref="Function2D.PlotType.Raster"/>, the resolution of the rasterised image on the X axis.</param>
00033 /// <param name="rasterResolutionY">If <paramref name="plotType"/> is <see
      cref="Function2D.PlotType.Raster"/>, the resolution of the rasterised image on the Y axis.</param>
00034 /// <param name="width">The width of the plot.</param>
00035 /// <param name="height">The height of the plot.</param>
```

```
00036 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00037 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00038 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00039 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00040 /// <param name="xAxisTitle">Title for the X axis.</param>
00041 /// <param name="yAxisTitle">Title for the Y axis.</param>
00042 /// <param name="title">Title for the plot.</param>
00043 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00044 /// <param name="pointElement">If <paramref name="plotType"/> is <see
      cref="Function2D.PlotType.SampledPoints"/>, the symbol drawn at the sampled function points.</param>
00045 /// <param name="colouring">Function used to associate function values to <see cref="Colour"/>s.  You
      should set this to a function accepting a single argument ranging from 0 to 1 and returning the
      corresponding <see cref="Colour"/>.  You can also use a <see cref="GradientStops"/> object.</param>
00046 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00047 /// <returns>A <see cref="Plot"/> containing the bar chart.</returns>
00048             public static Plot Function(Function2DGrid function, Function2D.PlotType plotType =
      Function2D.PlotType.Tessellation, int rasterResolutionX = 512, int rasterResolutionY = 512, double
      width = 350, double height = 250,
00049                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00050                 double axisArrowSize = 3,
00051                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00052                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00053                 string xAxisTitle = null,
00054                 string yAxisTitle = null,
00055                 string title = null,
00056                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00057                 IDataPointElement pointElement = null,
00058                 Func<double, Colour> colouring = null,
00059                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00060             {
00061
00062                 if (axisPresentationAttributes == null)
00063                 {
00064                     axisPresentationAttributes = new PlotElementPresentationAttributes();
00065                 }
00066
00067                 if (axisLabelPresentationAttributes == null)
00068                 {
00069                     axisLabelPresentationAttributes = new PlotElementPresentationAttributes() { Stroke
      = null };
00070                 }
00071
00072                 if (axisTitlePresentationAttributes == null)
00073                 {
00074                     axisTitlePresentationAttributes = new PlotElementPresentationAttributes() { Font =
      new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 14), Stroke =
      null };
00075                 }
00076
00077                 if (pointElement == null)
00078                 {
00079                     pointElement = new PathDataPointElement();
00080                 }
00081
00082                 if (coordinateSystem == null)
00083                 {
00084                     coordinateSystem = new LinearCoordinateSystem2D(function.DataPoints, width,
      height);
00085                 }
00086
00087                 if (titlePresentationAttributes == null)
00088                 {
00089                     titlePresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 18)
      };
00090                 }
00091
00092                 if (colouring == null)
00093                 {
00094                     colouring = Gradients.Viridis;
00095                 }
00096
00097                 (double minX, double minY, double maxX, double maxY, double rangeX, double rangeY) =
      GetDataRange(function.DataPoints);
00098
00099                 Point topLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, maxY });
00100                 Point topRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, maxY });
00101                 Point bottomRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, minY });
00102                 Point bottomLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, minY });
00103
00104                 double[] marginTopLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00105                 double[] marginTopRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
```

```
00106                 double[] marginBottomRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00107                 double[] marginBottomLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00108
00109                 double[] p1 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00110                 double[] p2 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00111                 double[] p3 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00112                 double[] p4 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00113
00114
00115                 double[] p5 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00116                 double[] p6 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00117                 double[] p7 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00118                 double[] p8 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00119
00120                 //Grid xGrid = new Grid(p1, p2, p3, p4, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00121                 //Grid yGrid = new Grid(p5, p6, p7, p8, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00122
00123                 ContinuousAxis xAxis = new ContinuousAxis(marginBottomLeft[0] < marginBottomRight[0] ?
      marginBottomLeft :  marginBottomRight, marginBottomLeft[0] < marginBottomRight[0] ?  marginBottomRight
      :  coordinateSystem.ToDataCoordinates(coordinateSystem.ToPlotCoordinates(marginBottomLeft) + new
      Point(-axisArrowSize - 7, 0)), coordinateSystem) { PresentationAttributes =
      axisPresentationAttributes, ArrowSize = axisArrowSize };
00124                 ContinuousAxis yAxis = new ContinuousAxis(marginBottomLeft[1] < marginTopLeft[1] ?
      marginBottomLeft :  marginTopLeft, marginBottomLeft[1] < marginTopLeft[1] ?  marginTopLeft :
      coordinateSystem.ToDataCoordinates(coordinateSystem.ToPlotCoordinates(marginBottomLeft) + new Point(0,
      axisArrowSize + 7)), coordinateSystem) { PresentationAttributes = axisPresentationAttributes,
      ArrowSize = axisArrowSize };
00125
00126                 ContinuousAxisTicks xTicks = new ContinuousAxisTicks(p3, p4, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00127                 ContinuousAxisTicks yTicks = new ContinuousAxisTicks(p6, p5, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00128
00129                 ContinuousAxisLabels xLabels = new ContinuousAxisLabels(p3, p4, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Alignment = TextAnchors.Center, Baseline =
      TextBaselines.Top, Rotation = 0, IntervalCount = 5 };
00130                 ContinuousAxisLabels yLabels = new ContinuousAxisLabels(p6, p5, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Position = _ => -10, Alignment =
      TextAnchors.Right, Rotation = 0, IntervalCount = 5 };
00131
00132                 Graphics xLabelsSize = new Graphics();
00133                 xLabels.Plot(xLabelsSize);
00134                 double xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
00135
00136                 Graphics yLabelsSize = new Graphics();
00137                 yLabels.Plot(yLabelsSize);
00138                 double yLabelsWidth = yLabelsSize.GetBounds().Size.Width;
00139
00140                 ContinuousAxisTitle xTitle = new ContinuousAxisTitle(xAxisTitle, marginBottomLeft,
      marginBottomRight, coordinateSystem, axisTitlePresentationAttributes) { Position = xLabelsHeight + 20,
      Alignment = TextAnchors.Center };
00141                 ContinuousAxisTitle yTitle = new ContinuousAxisTitle(yAxisTitle, marginBottomLeft,
      marginTopLeft, coordinateSystem, axisTitlePresentationAttributes) { Position = -20 - yLabelsWidth,
      Baseline = TextBaselines.Bottom, Alignment = TextAnchors.Center };
00142
00143                 TextLabel<IReadOnlyList<double» titleLabel = new
      TextLabel<IReadOnlyList<double»(title,
      coordinateSystem.ToDataCoordinates((coordinateSystem.ToPlotCoordinates(marginTopLeft) +
      coordinateSystem.ToPlotCoordinates(marginTopRight)) * 0.5 + new Point(0, -10)), coordinateSystem) {
      Baseline = TextBaselines.Bottom, PresentationAttributes = titlePresentationAttributes };
00144
00145                 Function2D functionPlot = new Function2D(function, coordinateSystem)
00146                 {
00147                     RasterResolutionX = rasterResolutionX,
00148                     RasterResolutionY = rasterResolutionY,
```

```
00149                   SampledPointElement = pointElement,
00150                   Type = plotType,
00151                   Colouring = colouring
00152               };
00153
00154               Plot tbr = new Plot();
00155               tbr.AddPlotElements(functionPlot, xAxis, yAxis, xTicks, yTicks, xLabels, yLabels,
      xTitle, yTitle);
00156
00157               tbr.AddPlotElement(titleLabel);
00158
00159               return tbr;
00160           }
00161
00162 /// <summary>
00163 /// Create a new plot for a function of two variables.
00164 /// </summary>
00165 /// <param name="function">The function to plot.</param>
00166 /// <param name="minX">The minimum value of the first function argument to plot.</param>
00167 /// <param name="minY">The minimum value of the second function argument to plot.</param>
00168 /// <param name="maxX">The maximum value of the first function argument to plot.</param>
00169 /// <param name="maxY">The maximum value of the second function argument to plot.</param>
00170 /// <param name="resolutionX">The distance between consecutive function samples on the X axis.</param>
00171 /// <param name="resolutionY">The distance between consecutive function samples on the Y axis.</param>
00172 /// <param name="gridType">The type of grid along which to sample the function.  If it is <see
      langword="null" />, this is determined automatically.</param>
00173 /// <param name="plotType">The type of plot to produce.</param>
00174 /// <param name="rasterResolutionX">If <paramref name="plotType"/> is <see
      cref="Function2D.PlotType.Raster"/>, the resolution of the rasterised image on the X axis.</param>
00175 /// <param name="rasterResolutionY">If <paramref name="plotType"/> is <see
      cref="Function2D.PlotType.Raster"/>, the resolution of the rasterised image on the Y axis.</param>
00176 /// <param name="width">The width of the plot.</param>
00177 /// <param name="height">The height of the plot.</param>
00178 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00179 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00180 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00181 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00182 /// <param name="xAxisTitle">Title for the X axis.</param>
00183 /// <param name="yAxisTitle">Title for the Y axis.</param>
00184 /// <param name="title">Title for the plot.</param>
00185 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00186 /// <param name="pointElement">If <paramref name="plotType"/> is <see
      cref="Function2D.PlotType.SampledPoints"/>, the symbol drawn at the sampled function points.</param>
00187 /// <param name="colouring">Function used to associate function values to <see cref="Colour"/>s.  You
      should set this to a function accepting a single argument ranging from 0 to 1 and returning the
      corresponding <see cref="Colour"/>.  You can also use a <see cref="GradientStops"/> object.</param>
00188 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00189 /// <returns>A <see cref="Plot"/> containing the bar chart.</returns>
00190           public static Plot Function(Func<double[], double> function, double minX, double minY,
      double maxX, double maxY, double resolutionX = double.NaN, double resolutionY = double.NaN,
      Function2DGrid.GridType?  gridType = null, Function2D.PlotType plotType =
      Function2D.PlotType.Tessellation, int rasterResolutionX = 512, int rasterResolutionY = 512, double
      width = 350, double height = 250,
00191               PlotElementPresentationAttributes axisPresentationAttributes = null,
00192               double axisArrowSize = 3,
00193               PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00194               PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00195               string xAxisTitle = null,
00196               string yAxisTitle = null,
00197               string title = null,
00198               PlotElementPresentationAttributes titlePresentationAttributes = null,
00199               IDataPointElement pointElement = null,
00200               Func<double, Colour> colouring = null,
00201               IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00202           {
00203               if (double.IsNaN(resolutionX) && double.IsNaN(resolutionY))
00204               {
00205                   double ratio = width / height;
00206
00207                   double stepsY = Math.Sqrt(4096 / ratio);
00208                   double stepsX = ratio * stepsY;
00209
00210                   resolutionX = (maxX - minX) / stepsX;
00211                   resolutionY = (maxY - minY) / stepsY;
00212               }
00213               else if (double.IsNaN(resolutionX) && !double.IsNaN(resolutionY))
00214               {
00215                   double ratio = width / height;
00216
00217                   double stepsY = (maxY - minY) / resolutionY;
00218                   double stepsX = ratio * stepsY;
00219
00220                   resolutionX = (maxX - minX) / stepsX;
00221               }
00222               else if (!double.IsNaN(resolutionX) && double.IsNaN(resolutionY))
00223               {
```

```
00224                     double ratio = width / height;
00225
00226                     double stepsX = (maxX - minX) / resolutionX;
00227                     double stepsY = stepsX / ratio;
00228
00229                     resolutionY = (maxY - minY) / stepsY;
00230                 }
00231
00232                 int stepsCountX = (int)Math.Floor((maxX - minX) / resolutionX);
00233                 int stepsCountY = (int)Math.Floor((maxY - minY) / resolutionY);
00234
00235                 if (gridType == null)
00236                 {
00237                     switch (plotType)
00238                     {
00239                         case Function2D.PlotType.SampledPoints:
00240                             gridType = Function2DGrid.GridType.HexagonHorizontal;
00241                             break;
00242
00243                         case Function2D.PlotType.Tessellation:
00244                             if ((stepsCountX + 1) * (stepsCountY + 1) <= 4096)
00245                             {
00246                                 gridType = Function2DGrid.GridType.Irregular;
00247                             }
00248                             else
00249                             {
00250                                 gridType = Function2DGrid.GridType.HexagonHorizontal;
00251                             }
00252                             break;
00253
00254                         case Function2D.PlotType.Raster:
00255                             gridType = Function2DGrid.GridType.Rectangular;
00256                             break;
00257                     }
00258                 }
00259
00260                 Function2DGrid grid = new Function2DGrid(function, minX, minY, maxX, maxY,
      stepsCountX, stepsCountY, gridType.Value);
00261
00262                 return Function(grid, plotType, rasterResolutionX, rasterResolutionY, width, height,
      axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      pointElement, colouring, coordinateSystem);
00263             }
00264
00265 /// <summary>
00266 /// Create a new plot for a function of two variables.
00267 /// </summary>
00268 /// <param name="function">The function to plot.</param>
00269 /// <param name="minX">The minimum value of the first function argument to plot.</param>
00270 /// <param name="minY">The minimum value of the second function argument to plot.</param>
00271 /// <param name="maxX">The maximum value of the first function argument to plot.</param>
00272 /// <param name="maxY">The maximum value of the second function argument to plot.</param>
00273 /// <param name="resolutionX">The distance between consecutive function samples on the X axis.</param>
00274 /// <param name="resolutionY">The distance between consecutive function samples on the Y axis.</param>
00275 /// <param name="gridType">The type of grid along which to sample the function.  If it is <see
      langword="null" />, this is determined automatically.</param>
00276 /// <param name="plotType">The type of plot to produce.</param>
00277 /// <param name="rasterResolutionX">If <paramref name="plotType"/> is <see
      cref="Function2D.PlotType.Raster"/>, the resolution of the rasterised image on the X axis.</param>
00278 /// <param name="rasterResolutionY">If <paramref name="plotType"/> is <see
      cref="Function2D.PlotType.Raster"/>, the resolution of the rasterised image on the Y axis.</param>
00279 /// <param name="width">The width of the plot.</param>
00280 /// <param name="height">The height of the plot.</param>
00281 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00282 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00283 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00284 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00285 /// <param name="xAxisTitle">Title for the X axis.</param>
00286 /// <param name="yAxisTitle">Title for the Y axis.</param>
00287 /// <param name="title">Title for the plot.</param>
00288 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00289 /// <param name="pointElement">If <paramref name="plotType"/> is <see
      cref="Function2D.PlotType.SampledPoints"/>, the symbol drawn at the sampled function points.</param>
00290 /// <param name="colouring">Function used to associate function values to <see cref="Colour"/>s.  You
      should set this to a function accepting a single argument ranging from 0 to 1 and returning the
      corresponding <see cref="Colour"/>.  You can also use a <see cref="GradientStops"/> object.</param>
00291 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00292 /// <returns>A <see cref="Plot"/> containing the bar chart.</returns>
00293             public static Plot Function(Func<double, double, double> function, double minX, double
      minY, double maxX, double maxY, double resolutionX = double.NaN, double resolutionY = double.NaN,
      Function2DGrid.GridType?  gridType = null, Function2D.PlotType plotType =
      Function2D.PlotType.Tessellation, int rasterResolutionX = 512, int rasterResolutionY = 512, double
      width = 350, double height = 250,
00294                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00295                 double axisArrowSize = 3,
```

```
00296                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00297                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00298                 string xAxisTitle = null,
00299                 string yAxisTitle = null,
00300                 string title = null,
00301                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00302                 IDataPointElement pointElement = null,
00303                 Func<double, Colour> colouring = null,
00304                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00305             {
00306                 return Function(x => function(x[0], x[1]), minX, minY, maxX, maxY, resolutionX,
      resolutionY, gridType, plotType, rasterResolutionX, rasterResolutionY, width, height,
      axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      pointElement, colouring, coordinateSystem);
00307             }
00308         }
00309     }
00310 }
```

8.49 Plot.Histogram.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021
00022 namespace VectSharp.Plots
00023 {
00024     partial class Plot
00025     {
00026 /// <summary>
00027 /// Describes the kind of normalisation that is performed.
00028 /// </summary>
00029         public enum NormalisationMode
00030         {
00031 /// <summary>
00032 /// No normalisation is performed.
00033 /// </summary>
00034             None,
00035
00036 /// <summary>
00037 /// Values are normalised so that the maximum of each distribution corresponds to the same value.
00038 /// </summary>
00039             Maximum,
00040
00041 /// <summary>
00042 /// Values are normalised so that the area covered by each distribution is the same.
00043 /// </summary>
00044             Area
00045         }
00046
00047         internal static (double q1, double q3, double iqr) IQR(IEnumerable<double> values)
00048         {
00049             double[] values2 = (from el in values orderby el ascending select el).ToArray();
00050
00051             double q1, q3;
00052
00053             if (values2.Length < 4)
00054             {
00055                 q1 = values2[0];
00056                 q3 = values2[values2.Length - 1];
00057             }
00058             else if (values2.Length == 4)
00059             {
00060                 q1 = (values2[0] + values2[1]) * 0.5;
00061                 q3 = (values2[2] + values2[3]) * 0.5;
```

```
00062             }
00063             else if (values2.Length % 2 == 0)
00064             {
00065                 q1 = values2.Length % 4 == 0 ?  (values2[values2.Length / 4] + values2[values2.Length
      / 4 + 1]) * 0.5 :  values2[values2.Length / 4];
00066                 q3 = values2.Length % 4 == 0 ?  (values2[3 * values2.Length / 4] + values2[3 *
      values2.Length / 4 + 1]) * 0.5 :  values2[3 * values2.Length / 4];
00067             }
00068             else if (values2.Length % 4 == 1)
00069             {
00070                 int n = (values2.Length - 1) / 4;
00071                 q1 = 0.75 * values2[n - 1] + 0.25 * values2[n];
00072                 q3 = 0.25 * values2[3 * n] + 0.75 * values2[3 * n + 1];
00073             }
00074             else
00075             {
00076                 int n = (values2.Length - 3) / 4;
00077                 q1 = 0.75 * values2[n] + 0.25 * values2[n + 1];
00078                 q3 = 0.25 * values2[3 * n + 1] + 0.75 * values2[3 * n + 2];
00079             }
00080
00081             return (q1, q3, q3 - q1);
00082         }
00083
00084         partial class Create
00085         {
00086             private static Plot Histogram(IReadOnlyList<IReadOnlyList<double» data, double
      overrideMax, double overrideMin, bool vertical = true, double margin = 0, double width = 350, double
      height = 250,
00087                     PlotElementPresentationAttributes axisPresentationAttributes = null,
00088                     double axisArrowSize = 3,
00089                     PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00090                     PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00091                     string xAxisTitle = null,
00092                     string yAxisTitle = null,
00093                     string title = null,
00094                     PlotElementPresentationAttributes titlePresentationAttributes = null,
00095                     PlotElementPresentationAttributes gridPresentationAttributes = null,
00096                     PlotElementPresentationAttributes dataPresentationAttributes = null,
00097                     IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00098             {
00099                 if (axisPresentationAttributes == null)
00100                 {
00101                     axisPresentationAttributes = new PlotElementPresentationAttributes();
00102                 }
00103
00104                 if (gridPresentationAttributes == null)
00105                 {
00106                     gridPresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      new SolidColourBrush(Colour.FromRgb(220, 220, 220)) };
00107                 }
00108
00109                 if (axisLabelPresentationAttributes == null)
00110                 {
00111                     axisLabelPresentationAttributes = new PlotElementPresentationAttributes() { Stroke
      = null };
00112                 }
00113
00114                 if (axisTitlePresentationAttributes == null)
00115                 {
00116                     axisTitlePresentationAttributes = new PlotElementPresentationAttributes() { Font =
      new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 14), Stroke =
      null };
00117                 }
00118
00119                 if (dataPresentationAttributes == null)
00120                 {
00121                     dataPresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Fill = new SolidColourBrush(Colour.FromRgb(0, 114, 178)) };
00122                 }
00123
00124                 double baselineValue = 0;
00125
00126                 if (coordinateSystem is LogarithmicCoordinateSystem2D || (!vertical &&
      coordinateSystem is LinLogCoordinateSystem2D) || (vertical && coordinateSystem is
      LogLinCoordinateSystem2D))
00127                 {
00128                     baselineValue = 1;
00129                 }
00130
00131                 Bars bars;
00132
00133                 if (baselineValue == 0)
00134                 {
00135                     bars = new Bars(data, coordinateSystem, vertical) { PresentationAttributes =
      dataPresentationAttributes, Margin = margin };
00136                 }
```

```
00137                 else
00138                 {
00139                     bars = new Bars(data, new Comparison<IReadOnlyList<double»(vertical ?  new
      Func<IReadOnlyList<double>, IReadOnlyList<double>, int>((x, y) => Math.Sign(x[0] - y[0])) :  (x, y) =>
      Math.Sign(x[1] - y[1])), vertical ?  new Func<IReadOnlyList<double>, IReadOnlyList<double»(pt => new
      double[] { pt[0], baselineValue }) :  pt => new double[] { baselineValue, pt[1] }, coordinateSystem) {
      PresentationAttributes = dataPresentationAttributes, Margin = margin };
00140                 }
00141
00142                 (double minX, double minY, double maxX, double maxY, double rangeX, double rangeY) =
      GetDataRange(data);
00143
00144                 double dataMinX = minX;
00145                 double dataMinY = minY;
00146                 double dataMaxX = maxX;
00147                 double dataMaxY = maxY;
00148
00149                 if (coordinateSystem == null)
00150                 {
00151                     LinearCoordinateSystem2D linCoords = new LinearCoordinateSystem2D(data, width,
      height);
00152
00153                     if (vertical)
00154                     {
00155                         minY = Math.Min(minY, 0);
00156                         maxY = Math.Max(maxY, 0);
00157
00158                         if (!double.IsNaN(overrideMax))
00159                         {
00160                             maxY = overrideMax;
00161                         }
00162
00163                         if (!double.IsNaN(overrideMin))
00164                         {
00165                             minY = overrideMin;
00166                         }
00167
00168                         rangeY = maxY - minY;
00169
00170                         linCoords.MinY = minY - rangeY * 0.1;
00171                         linCoords.MaxY = maxY + rangeY * 0.1;
00172
00173
00174                         if (bars.Data.Count >= 2)
00175                         {
00176                             IReadOnlyList<double> item0 = bars.Data.ElementAt(0);
00177                             IReadOnlyList<double> item1 = bars.Data.ElementAt(1);
00178
00179                             IReadOnlyList<double> itemN = bars.Data.ElementAt(bars.Data.Count - 1);
00180                             IReadOnlyList<double> itemN1 = bars.Data.ElementAt(bars.Data.Count - 2);
00181
00182                             minX = Math.Min(minX, 1.5 * item0[0] - item1[0] * 0.5);
00183                             maxX = Math.Max(maxX, 1.5 * itemN[0] - itemN1[0] * 0.5);
00184
00185                             rangeX = maxX - minX;
00186                             linCoords.MinX = minX;
00187                             linCoords.MaxX = maxX;
00188                         }
00189                     }
00190                     else
00191                     {
00192                         minX = Math.Min(minX, 0);
00193                         maxX = Math.Max(maxX, 0);
00194
00195                         if (!double.IsNaN(overrideMax))
00196                         {
00197                             maxX = overrideMax;
00198                         }
00199
00200                         if (!double.IsNaN(overrideMin))
00201                         {
00202                             minX = overrideMin;
00203                         }
00204
00205                         rangeX = maxX - minX;
00206
00207                         linCoords.MinX = minX - rangeX * 0.1;
00208                         linCoords.MaxX = maxX + rangeX * 0.1;
00209
00210
00211                         if (bars.Data.Count >= 2)
00212                         {
00213                             IReadOnlyList<double> item0 = bars.Data.ElementAt(0);
00214                             IReadOnlyList<double> item1 = bars.Data.ElementAt(1);
00215
00216                             IReadOnlyList<double> itemN = bars.Data.ElementAt(bars.Data.Count - 1);
00217                             IReadOnlyList<double> itemN1 = bars.Data.ElementAt(bars.Data.Count - 2);
```

```
00218
00219                           minY = Math.Min(minY, 1.5 * item0[1] - item1[1] * 0.5);
00220                           maxY = Math.Max(maxY, 1.5 * itemN[1] - itemN1[1] * 0.5);
00221
00222                           rangeY = maxY - minY;
00223                           linCoords.MinY = minY;
00224                           linCoords.MaxY = maxY;
00225                       }
00226                   }
00227
00228                   coordinateSystem = linCoords;
00229                   bars.CoordinateSystem = coordinateSystem;
00230               }
00231               else
00232               {
00233                   if (vertical)
00234                   {
00235                       minY = Math.Min(minY, baselineValue);
00236                       rangeY = maxY - minY;
00237
00238                       if (bars.Data.Count >= 2)
00239                       {
00240                           IReadOnlyList<double> item0 = bars.Data.ElementAt(0);
00241                           IReadOnlyList<double> item1 = bars.Data.ElementAt(1);
00242
00243                           IReadOnlyList<double> itemN = bars.Data.ElementAt(bars.Data.Count - 1);
00244                           IReadOnlyList<double> itemN1 = bars.Data.ElementAt(bars.Data.Count - 2);
00245
00246                           minX = Math.Min(minX, 1.5 * item0[0] - item1[0] * 0.5);
00247                           maxX = Math.Max(maxX, 1.5 * itemN[0] - itemN1[0] * 0.5);
00248
00249                           rangeX = maxX - minX;
00250                       }
00251                   }
00252                   else
00253                   {
00254                       minX = Math.Min(minX, baselineValue);
00255                       rangeX = maxX - minX;
00256
00257                       if (bars.Data.Count >= 2)
00258                       {
00259                           IReadOnlyList<double> item0 = bars.Data.ElementAt(0);
00260                           IReadOnlyList<double> item1 = bars.Data.ElementAt(1);
00261
00262                           IReadOnlyList<double> itemN = bars.Data.ElementAt(bars.Data.Count - 1);
00263                           IReadOnlyList<double> itemN1 = bars.Data.ElementAt(bars.Data.Count - 2);
00264
00265                           minY = Math.Min(minY, 1.5 * item0[1] - item1[1] * 0.5);
00266                           maxY = Math.Max(maxY, 1.5 * itemN[1] - itemN1[1] * 0.5);
00267
00268                           rangeY = maxY - minY;
00269                       }
00270                   }
00271               }
00272
00273
00274
00275               if (titlePresentationAttributes == null)
00276               {
00277                   titlePresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 18)
      };
00278               }
00279
00280               Point topLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, maxY });
00281               Point topRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, maxY });
00282               Point bottomRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, minY });
00283               Point bottomLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, minY });
00284
00285               double[] marginTopLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00286               double[] marginTopRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00287               double[] marginBottomRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00288               double[] marginBottomLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00289
00290               double[] p1 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00291               double[] p2 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
```

```
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00292                 double[] p3 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00293                 double[] p4 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00294
00295
00296                 double[] p5 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00297                 double[] p6 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00298                 double[] p7 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00299                 double[] p8 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00300
00301                 if (vertical)
00302                 {
00303                     Point p1p = new Point(Math.Min(Math.Min(topLeft.X, topRight.X),
      Math.Min(bottomLeft.X, bottomRight.X)), Math.Min(Math.Min(topLeft.Y, topRight.Y),
      Math.Min(bottomLeft.Y, bottomRight.Y)) - 10);
00304                     Point p1p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMinX, dataMaxY
      });
00305
00306                     Point p2p = new Point(Math.Max(Math.Max(topLeft.X, topRight.X),
      Math.Max(bottomLeft.X, bottomRight.X)), Math.Min(Math.Min(topLeft.Y, topRight.Y),
      Math.Min(bottomLeft.Y, bottomRight.Y)) - 10);
00307                     Point p2p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMaxX, dataMaxY
      });
00308
00309                     Point p3p = new Point(Math.Min(Math.Min(topLeft.X, topRight.X),
      Math.Min(bottomLeft.X, bottomRight.X)), Math.Max(Math.Max(topLeft.Y, topRight.Y),
      Math.Max(bottomLeft.Y, bottomRight.Y)) + 10);
00310                     Point p3p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMinX, 0 });
00311
00312                     Point p4p = new Point(Math.Max(Math.Max(topLeft.X, topRight.X),
      Math.Max(bottomLeft.X, bottomRight.X)), Math.Max(Math.Max(topLeft.Y, topRight.Y),
      Math.Max(bottomLeft.Y, bottomRight.Y)) + 10);
00313                     Point p4p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMaxX, 0 });
00314
00315                     p1 = coordinateSystem.ToDataCoordinates(new Point(p1p2.X, p1p.Y));
00316                     p2 = coordinateSystem.ToDataCoordinates(new Point(p2p2.X, p2p.Y));
00317                     p3 = coordinateSystem.ToDataCoordinates(new Point(p3p2.X, p3p.Y));
00318                     p4 = coordinateSystem.ToDataCoordinates(new Point(p4p2.X, p4p.Y));
00319                 }
00320                 else
00321                 {
00322                     Point p5p = new Point(Math.Min(Math.Min(topLeft.X, topRight.X),
      Math.Min(bottomLeft.X, bottomRight.X)) - 10, Math.Min(Math.Min(topLeft.Y, topRight.Y),
      Math.Min(bottomLeft.Y, bottomRight.Y)));
00323                     Point p5p2 = coordinateSystem.ToPlotCoordinates(new double[] { 0, dataMaxY });
00324
00325                     Point p6p = new Point(Math.Min(Math.Min(topLeft.X, topRight.X),
      Math.Min(bottomLeft.X, bottomRight.X)) - 10, Math.Max(Math.Max(topLeft.Y, topRight.Y),
      Math.Max(bottomLeft.Y, bottomRight.Y)));
00326                     Point p6p2 = coordinateSystem.ToPlotCoordinates(new double[] { 0, dataMinY });
00327
00328                     Point p7p = new Point(Math.Max(Math.Max(topLeft.X, topRight.X),
      Math.Max(bottomLeft.X, bottomRight.X)) + 10, Math.Min(Math.Min(topLeft.Y, topRight.Y),
      Math.Min(bottomLeft.Y, bottomRight.Y)));
00329                     Point p7p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMaxX, dataMaxY
      });
00330
00331                     Point p8p = new Point(Math.Max(Math.Max(topLeft.X, topRight.X),
      Math.Max(bottomLeft.X, bottomRight.X)) + 10, Math.Max(Math.Max(topLeft.Y, topRight.Y),
      Math.Max(bottomLeft.Y, bottomRight.Y)));
00332                     Point p8p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMinX, dataMinY
      });
00333
00334                     p5 = coordinateSystem.ToDataCoordinates(new Point(p5p.X, p5p2.Y));
00335                     p6 = coordinateSystem.ToDataCoordinates(new Point(p6p.X, p6p2.Y));
00336                     p7 = coordinateSystem.ToDataCoordinates(new Point(p7p2.X, p7p.Y));
00337                     p8 = coordinateSystem.ToDataCoordinates(new Point(p8p2.X, p8p.Y));
00338                 }
00339
00340                 Grid grid;
00341
00342                 if (vertical)
00343                 {
00344                     grid = new Grid(p5, p6, p7, p8, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
```

```
00345                 }
00346                 else
00347                 {
00348                     grid = new Grid(p1, p2, p3, p4, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00349                 }
00350 
00351                 ContinuousAxis xAxis = new ContinuousAxis(marginBottomLeft[0] < marginBottomRight[0] ?
      marginBottomLeft :  marginBottomRight, marginBottomLeft[0] < marginBottomRight[0] ?  marginBottomRight
      :  vertical ?  marginBottomLeft :
      coordinateSystem.ToDataCoordinates(coordinateSystem.ToPlotCoordinates(marginBottomLeft) + new
      Point(-axisArrowSize - 7, 0)), coordinateSystem) { PresentationAttributes =
      axisPresentationAttributes, ArrowSize = axisArrowSize };
00352                 ContinuousAxis yAxis = new ContinuousAxis(marginBottomLeft[1] < marginTopLeft[1] ?
      marginBottomLeft :  marginTopLeft, marginBottomLeft[1] < marginTopLeft[1] ?  marginTopLeft :
      !vertical ?  marginBottomLeft :
      coordinateSystem.ToDataCoordinates(coordinateSystem.ToPlotCoordinates(marginBottomLeft) + new Point(0,
      axisArrowSize + 7)), coordinateSystem) { PresentationAttributes = axisPresentationAttributes,
      ArrowSize = axisArrowSize };
00353 
00354                 int every = (int)Math.Ceiling((double)data.Count / 5);
00355                 int shift = (data.Count - every * (data.Count / every - 1) - 1) / 2;
00356 
00357                 ContinuousAxisTicks xTicks;
00358                 ContinuousAxisTicks yTicks;
00359 
00360                 if (vertical)
00361                 {
00362                     xTicks = new ContinuousAxisTicks(p3, p4, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes, IntervalCount = data.Count - 1, SizeAbove = i =>
      (i - shift) % every == 0 ?  3 :  2, SizeBelow = i => (i - shift) % every == 0 ?  3 :  2 };
00363                     yTicks = new ContinuousAxisTicks(p6, p5, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00364                 }
00365                 else
00366                 {
00367                     xTicks = new ContinuousAxisTicks(p3, p4, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00368                     yTicks = new ContinuousAxisTicks(p6, p5, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes, IntervalCount = data.Count - 1, SizeAbove = i =>
      (i - shift) % every == 0 ?  3 :  2, SizeBelow = i => (i - shift) % every == 0 ?  3 :  2 };
00369                 }
00370 
00371                 ContinuousAxisLabels xLabels;
00372                 ContinuousAxisLabels yLabels;
00373 
00374                 if (vertical)
00375                 {
00376                     xLabels = new ContinuousAxisLabels(p3, p4, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Alignment = TextAnchors.Center, Baseline =
      TextBaselines.Top, Rotation = 0, IntervalCount = data.Count - 1 };
00377                     yLabels = new ContinuousAxisLabels(p6, p5, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Position = _ => -10, Alignment =
      TextAnchors.Right, Rotation = 0, IntervalCount = 5 };
00378 
00379                     Func<IReadOnlyList<double>, int, IEnumerable<FormattedText» originalFormatter =
      xLabels.TextFormat;
00380                     xLabels.TextFormat = (x, i) => (i - shift) % every == 0 ?  originalFormatter(x, i)
      :  null;
00381                 }
00382                 else
00383                 {
00384                     xLabels = new ContinuousAxisLabels(p3, p4, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Alignment = TextAnchors.Center, Baseline =
      TextBaselines.Top, Rotation = 0, IntervalCount = 5 };
00385                     yLabels = new ContinuousAxisLabels(p6, p5, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Position = _ => -10, Alignment =
      TextAnchors.Right, Rotation = 0, IntervalCount = data.Count - 1 };
00386 
00387                     Func<IReadOnlyList<double>, int, IEnumerable<FormattedText» originalFormatter =
      yLabels.TextFormat;
00388                     yLabels.TextFormat = (x, i) => (i - shift) % every == 0 ?  originalFormatter(x, i)
      :  null;
00389                 }
00390 
00391                 Graphics xLabelsSize = new Graphics();
00392                 xLabels.Plot(xLabelsSize);
00393                 double xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
00394 
00395                 Graphics yLabelsSize = new Graphics();
00396                 yLabels.Plot(yLabelsSize);
00397                 double yLabelsWidth = yLabelsSize.GetBounds().Size.Width;
00398 
00399                 ContinuousAxisTitle xTitle = new ContinuousAxisTitle(xAxisTitle, marginBottomLeft,
      marginBottomRight, coordinateSystem, axisTitlePresentationAttributes) { Position = xLabelsHeight + 20,
      Alignment = TextAnchors.Center };
00400                 ContinuousAxisTitle yTitle = new ContinuousAxisTitle(yAxisTitle, marginBottomLeft,
```

```
      marginTopLeft, coordinateSystem, axisTitlePresentationAttributes) { Position = -20 - yLabelsWidth,
      Baseline = TextBaselines.Bottom, Alignment = TextAnchors.Center };
00401
00402                 TextLabel<IReadOnlyList<double» titleLabel = new
      TextLabel<IReadOnlyList<double»(title,
      coordinateSystem.ToDataCoordinates((coordinateSystem.ToPlotCoordinates(marginTopLeft) +
      coordinateSystem.ToPlotCoordinates(marginTopRight)) * 0.5 + new Point(0, -10)), coordinateSystem) {
      Baseline = TextBaselines.Bottom, PresentationAttributes = titlePresentationAttributes };
00403
00404                 Plot tbr = new Plot();
00405                 tbr.AddPlotElements(grid, xAxis, yAxis, xTicks, yTicks, xLabels, yLabels, xTitle,
      yTitle, bars, titleLabel);
00406
00407                 return tbr;
00408             }
00409
00410 /// <summary>
00411 /// Create a new histogram.
00412 /// </summary>
00413 /// <param name="data">The data to plot.</param>
00414 /// <param name="vertical">If this is <see langword="true"/> (the default), the bars go from the X
      axis up to the sampled values.  If this is <see langword="false"/>, the bars go from the Y axis to the
      sampled values.</param>
00415 /// <param name="margin">Spacing between consecutive bars.  This should be a value between 0 and
      1.</param>
00416 /// <param name="width">The width of the plot.</param>
00417 /// <param name="height">The height of the plot.</param>
00418 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00419 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00420 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00421 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00422 /// <param name="xAxisTitle">Title for the X axis.</param>
00423 /// <param name="yAxisTitle">Title for the Y axis.</param>
00424 /// <param name="title">Title for the plot.</param>
00425 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00426 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00427 /// <param name="dataPresentationAttributes">Presentation attributes for the plotted data.</param>
00428 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00429 /// <returns>A <see cref="Plot"/> containing the histogram.</returns>
00430             public static Plot Histogram(IReadOnlyList<IReadOnlyList<double» data, bool vertical =
      true, double margin = 0, double width = 350, double height = 250,
00431                     PlotElementPresentationAttributes axisPresentationAttributes = null,
00432                     double axisArrowSize = 3,
00433                     PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00434                     PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00435                     string xAxisTitle = null,
00436                     string yAxisTitle = null,
00437                     string title = null,
00438                     PlotElementPresentationAttributes titlePresentationAttributes = null,
00439                     PlotElementPresentationAttributes gridPresentationAttributes = null,
00440                     PlotElementPresentationAttributes dataPresentationAttributes = null,
00441                     IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00442             {
00443                 return Histogram(data, double.NaN, double.NaN, vertical, margin, width, height,
      axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, dataPresentationAttributes, coordinateSystem);
00444             }
00445
00446 /// <summary>
00447 /// Create a new histogram.
00448 /// </summary>
00449 /// <param name="data">The data whose distribution will be plotted.</param>
00450 /// <param name="vertical">If this is <see langword="true"/> (the default), the bars go from the X
      axis up to the sampled values.  If this is <see langword="false"/>, the bars go from the Y axis to the
      sampled values.</param>
00451 /// <param name="margin">Spacing between consecutive bars.  This should be a value between 0 and
      1.</param>
00452 /// <param name="binCount">The number of bins to use.  If this is &lt; 2, the number of bins is
      determined automatically using the Freedman-Diaconis rule.</param>
00453 /// <param name="underflow">Values smaller than this will be excluded from the main plot and moved
      into an underflow bin.</param>
00454 /// <param name="overflow">Values larger than this will be excluded from the main plot and moved into
      an overflow bin.</param>
00455 /// <param name="width">The width of the plot.</param>
00456 /// <param name="height">The height of the plot.</param>
00457 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00458 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00459 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00460 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00461 /// <param name="xAxisTitle">Title for the X axis.</param>
00462 /// <param name="yAxisTitle">Title for the Y axis.</param>
00463 /// <param name="title">Title for the plot.</param>
00464 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00465 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00466 /// <param name="dataPresentationAttributes">Presentation attributes for the plotted data.</param>
```

```
00467 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00468 /// <returns>A <see cref="Plot"/> containing the histogram.</returns>
00469             public static Plot Histogram(IReadOnlyList<double> data, bool vertical = true, double
      margin = 0, int binCount = -1, double underflow = double.NegativeInfinity, double overflow =
      double.PositiveInfinity, double width = 350, double height = 250,
00470                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00471                 double axisArrowSize = 3,
00472                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00473                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00474                 string xAxisTitle = null,
00475                 string yAxisTitle = null,
00476                 string title = null,
00477                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00478                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00479                 PlotElementPresentationAttributes dataPresentationAttributes = null,
00480                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00481             {
00482                 if (dataPresentationAttributes == null)
00483                 {
00484                     dataPresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      new SolidColourBrush(Colour.FromRgb(0, 114, 178)), LineWidth = 0.5, Fill = new
      SolidColourBrush(Colour.FromRgb(0, 114, 178)) };
00485                 }
00486 
00487                 int underflowCount = 0;
00488                 int overflowCount = 0;
00489 
00490                 List<double> binnableData = new List<double>(data.Count);
00491 
00492                 double min = double.MaxValue;
00493                 double max = double.MinValue;
00494 
00495                 for (int i = 0; i < data.Count; i++)
00496                 {
00497                     if (data[i] > underflow && data[i] < overflow)
00498                     {
00499                         min = Math.Min(min, data[i]);
00500                         max = Math.Max(max, data[i]);
00501                         binnableData.Add(data[i]);
00502                     }
00503                     else if (data[i] <= underflow)
00504                     {
00505                         underflowCount++;
00506                     }
00507                     else if (data[i] >= overflow)
00508                     {
00509                         overflowCount++;
00510                     }
00511                 }
00512 
00513                 if (binCount <= 1)
00514                 {
00515                     (double _, double _, double iqr) = IQR(binnableData);
00516                     double h2 = 2 * iqr / Math.Pow(binnableData.Count, 1.0 / 3.0);
00517 
00518                     if (h2 > 0)
00519                     {
00520                         binCount = Math.Max(2, (int)Math.Ceiling((max - min) / h2));
00521                     }
00522                     else
00523                     {
00524                         binCount = 2;
00525                     }
00526                 }
00527 
00528                 int[] bins = new int[binCount];
00529 
00530                 if (max > min)
00531                 {
00532                     for (int i = 0; i < binnableData.Count; i++)
00533                     {
00534                         int index = (int)Math.Min(binCount - 1, Math.Floor((binnableData[i] - min) /
      (max - min) * binCount));
00535 
00536                         bins[index]++;
00537                     }
00538                 }
00539                 else
00540                 {
00541                     bins[0] = binnableData.Count;
00542                 }
00543 
00544                 (string, double)[] binnedData = new (string, double)[bins.Length];
00545 
00546                 for (int i = 0; i < bins.Length; i++)
00547                 {
```

```
00548                     double binStart = min + (max - min) / binCount * i;
00549                     double binEnd = min + (max - min) / binCount * (i + 1);
00550 
00551                     string formatString;
00552 
00553                     if (binEnd - binStart >= 10)
00554                     {
00555                         formatString = "0";
00556                     }
00557                     else if (binEnd - binStart >= 1)
00558                     {
00559                         formatString = "0.0";
00560                     }
00561                     else if (binEnd == binStart)
00562                     {
00563                         formatString = "0." + new string('0',
    -(int)Math.Floor(Math.Log10(Math.Abs(binEnd))) + 1);
00564                     }
00565                     else
00566                     {
00567                         formatString = "0." + new string('0', -(int)Math.Floor(Math.Log10(binEnd -
    binStart)) + 1);
00568                     }
00569 
00570                     string binLabel = "[" + binStart.ToString(formatString,
    System.Globalization.CultureInfo.InvariantCulture) + ", " + binEnd.ToString(formatString,
    System.Globalization.CultureInfo.InvariantCulture);
00571 
00572                     if (i < bins.Length - 1)
00573                     {
00574                         binLabel += ")";
00575                     }
00576                     else
00577                     {
00578                         binLabel += "]";
00579                     }
00580 
00581                     binnedData[i] = (binLabel, bins[i]);
00582                 }
00583 
00584                 double maxY = Math.Max(Math.Max(underflowCount, overflowCount), bins.Max());
00585 
00586                 Plot plot = Create.BarChart(binnedData, maxY, double.NaN, vertical, margin, width,
    height, axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
    axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
    gridPresentationAttributes, dataPresentationAttributes, coordinateSystem);
00587 
00588                 if (vertical)
00589                 {
00590                     DataLabels<IReadOnlyList<double» xLabels =
    plot.GetFirst<DataLabels<IReadOnlyList<double»>();
00591 
00592                     xLabels.Alignment = TextAnchors.Left;
00593 
00594                     xLabels.Rotation = (a, b) => Math.PI / 6;
00595 
00596                     xLabels.Baseline = TextBaselines.Middle;
00597 
00598                     Func<int, IReadOnlyList<double>, object> prevLabels = xLabels.Label;
00599 
00600                     xLabels.Label = (i, x) =>
00601                     {
00602                         if (i == 0 || i == bins.Length - 1 || i == Math.Floor(bins.Length * 0.5) || i
    == Math.Floor(bins.Length * 0.25) || i == Math.Floor(bins.Length * 0.75))
00603                         {
00604                             return prevLabels(i, x);
00605                         }
00606                         else
00607                         {
00608                             return null;
00609                         }
00610                     };
00611 
00612                     plot.GetFirst<ContinuousAxisTicks>().SizeAbove = i => (i == 0 || i == bins.Length
    - 1 || i == Math.Floor(bins.Length * 0.5) || i == Math.Floor(bins.Length * 0.25) || i ==
    Math.Floor(bins.Length * 0.75)) ?  3 :  2;
00613                     plot.GetFirst<ContinuousAxisTicks>().SizeBelow = i => (i == 0 || i == bins.Length
    - 1 || i == Math.Floor(bins.Length * 0.5) || i == Math.Floor(bins.Length * 0.25) || i ==
    Math.Floor(bins.Length * 0.75)) ?  3 :  2;
00614 
00615                     Graphics xLabelsSize = new Graphics();
00616                     xLabels.Plot(xLabelsSize);
00617                     double xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
00618 
00619                     plot.GetFirst<ContinuousAxisTitle>().Position = xLabelsHeight + 20;
00620 
00621                 }
```

```
00622                 else
00623                 {
00624                     DataLabels<IReadOnlyList<double» xLabels =
      plot.GetFirst<DataLabels<IReadOnlyList<double»>();
00625 
00626                     Func<int, IReadOnlyList<double>, object> prevLabels = xLabels.Label;
00627 
00628                     xLabels.Label = (i, x) =>
00629                     {
00630                         if (i == 0 || i == bins.Length - 1 || i == Math.Floor(bins.Length * 0.5) || i
      == Math.Floor(bins.Length * 0.25) || i == Math.Floor(bins.Length * 0.75))
00631                         {
00632                             return prevLabels(i, x);
00633                         }
00634                         else
00635                         {
00636                             return null;
00637                         }
00638                     };
00639 
00640                     plot.GetAll<ContinuousAxisTicks>().ElementAt(1).SizeAbove = i => (i == 0 || i ==
      bins.Length - 1 || i == Math.Floor(bins.Length * 0.5) || i == Math.Floor(bins.Length * 0.25) || i ==
      Math.Floor(bins.Length * 0.75)) ?  3 :  2;
00641                     plot.GetAll<ContinuousAxisTicks>().ElementAt(1).SizeBelow = i => (i == 0 || i ==
      bins.Length - 1 || i == Math.Floor(bins.Length * 0.5) || i == Math.Floor(bins.Length * 0.25) || i ==
      Math.Floor(bins.Length * 0.75)) ?  3 :  2;
00642                 }
00643 
00644                 if (underflowCount > 0)
00645                 {
00646                     if (vertical)
00647                     {
00648                         ContinuousAxis xAxis = plot.GetFirst<ContinuousAxis>();
00649                         xAxis.StartPoint = new double[] { xAxis.StartPoint[0] - 1.5,
      xAxis.StartPoint[1] };
00650 
00651                         ContinuousAxis yAxis = plot.GetAll<ContinuousAxis>().ElementAt(1);
00652 
00653                         yAxis.StartPoint = new double[] { yAxis.StartPoint[0] - 1.5,
      yAxis.StartPoint[1] };
00654                         yAxis.EndPoint = new double[] { yAxis.EndPoint[0] - 1.5, yAxis.EndPoint[1] };
00655 
00656                         Grid grid = plot.GetFirst<Grid>();
00657                         grid.Side1Start = new double[] { grid.Side1Start[0] - 1.5, grid.Side1Start[1]
      };
00658                         grid.Side1End = new double[] { grid.Side1End[0] - 1.5, grid.Side1End[1] };
00659 
00660                         ContinuousAxisTicks yTicks = plot.GetAll<ContinuousAxisTicks>().ElementAt(1);
00661                         yTicks.StartPoint = new double[] { yTicks.StartPoint[0] - 1.5,
      yTicks.StartPoint[1] };
00662                         yTicks.EndPoint = new double[] { yTicks.EndPoint[0] - 1.5, yTicks.EndPoint[1]
      };
00663 
00664                         ContinuousAxisLabels yLabels = plot.GetFirst<ContinuousAxisLabels>();
00665                         yLabels.StartPoint = new double[] { yLabels.StartPoint[0] - 1.5,
      yLabels.StartPoint[1] };
00666                         yLabels.EndPoint = new double[] { yLabels.EndPoint[0] - 1.5,
      yLabels.EndPoint[1] };
00667 
00668                         ContinuousAxisTitle yTitle = plot.GetAll<ContinuousAxisTitle>().ElementAt(1);
00669                         yTitle.StartPoint = yAxis.StartPoint;
00670                         yTitle.EndPoint = yAxis.EndPoint;
00671 
00672                     }
00673                     else
00674                     {
00675                         ContinuousAxis xAxis = plot.GetAll<ContinuousAxis>().ElementAt(1);
00676                         xAxis.StartPoint = new double[] { xAxis.StartPoint[0], xAxis.StartPoint[1] -
      1.5 };
00677 
00678                         ContinuousAxis yAxis = plot.GetFirst<ContinuousAxis>();
00679 
00680                         yAxis.StartPoint = new double[] { yAxis.StartPoint[0], yAxis.StartPoint[1] -
      1.5 };
00681                         yAxis.EndPoint = new double[] { yAxis.EndPoint[0], yAxis.EndPoint[1] - 1.5 };
00682 
00683                         Grid grid = plot.GetFirst<Grid>();
00684                         grid.Side2Start = new double[] { grid.Side2Start[0], grid.Side2Start[1] - 1.5
      };
00685                         grid.Side2End = new double[] { grid.Side2End[0], grid.Side2End[1] - 1.5 };
00686 
00687                         ContinuousAxisTicks yTicks = plot.GetFirst<ContinuousAxisTicks>();
00688                         yTicks.StartPoint = new double[] { yTicks.StartPoint[0], yTicks.StartPoint[1]
      - 1.5 };
00689                         yTicks.EndPoint = new double[] { yTicks.EndPoint[0], yTicks.EndPoint[1] - 1.5
      };
00690 
```

```
00691                           ContinuousAxisLabels yLabels = plot.GetFirst<ContinuousAxisLabels>();
00692                           yLabels.StartPoint = new double[] { yLabels.StartPoint[0],
    yLabels.StartPoint[1] - 1.5 };
00693                           yLabels.EndPoint = new double[] { yLabels.EndPoint[0], yLabels.EndPoint[1] -
    1.5 };
00694
00695                           ContinuousAxisTitle yTitle = plot.GetFirst<ContinuousAxisTitle>();
00696                           yTitle.StartPoint = yAxis.StartPoint;
00697                           yTitle.EndPoint = yAxis.EndPoint;
00698                       }
00699                   }
00700
00701                   if (overflowCount > 0)
00702                   {
00703                       if (vertical)
00704                       {
00705                           ContinuousAxis xAxis = plot.GetFirst<ContinuousAxis>();
00706                           xAxis.EndPoint = new double[] { xAxis.EndPoint[0] + 1.5, xAxis.EndPoint[1] };
00707
00708                           Grid grid = plot.GetFirst<Grid>();
00709                           grid.Side2Start = new double[] { grid.Side2Start[0] + 1.5, grid.Side2Start[1]
    };
00710                           grid.Side2End = new double[] { grid.Side2End[0] + 1.5, grid.Side2End[1] };
00711                       }
00712                       else
00713                       {
00714                           ContinuousAxis xAxis = plot.GetAll<ContinuousAxis>().ElementAt(1);
00715                           xAxis.EndPoint = new double[] { xAxis.EndPoint[0], xAxis.EndPoint[1] + 1.5 };
00716
00717                           Grid grid = plot.GetFirst<Grid>();
00718                           grid.Side1Start = new double[] { grid.Side1Start[0], grid.Side1Start[1] + 1.5
    };
00719                           grid.Side1End = new double[] { grid.Side1End[0], grid.Side1End[1] + 1.5 };
00720
00721                           TextLabel<IReadOnlyList<double» titleLabel =
    plot.GetFirst<TextLabel<IReadOnlyList<double»>();
00722                           titleLabel.Position = new double[] { titleLabel.Position[0],
    titleLabel.Position[1] + 1.5 };
00723                       }
00724                   }
00725
00726                   if (overflowCount > 0 || underflowCount > 0)
00727                   {
00728                       DataLabels<IReadOnlyList<double» xLabels =
    plot.GetFirst<DataLabels<IReadOnlyList<double»>();
00729
00730                       List<IReadOnlyList<double» xLabelData = xLabels.Data.ToList();
00731
00732                       List<IReadOnlyList<double» barData = new List<IReadOnlyList<double»();
00733
00734                       if (underflowCount > 0)
00735                       {
00736                           if (vertical)
00737                           {
00738                               xLabelData.Add(plot.GetFirst<Bars>().GetBaseline(new double[] { -1.5, 0
    }));
00739
00740                               barData.Add(new double[] { -1.5, underflowCount });
00741                               barData.Add(plot.GetFirst<Bars>().GetBaseline(new double[] { -0.5, 0 }));
00742                           }
00743                           else
00744                           {
00745                               xLabelData.Add(plot.GetFirst<Bars>().GetBaseline(new double[] { 0, -1.5
    }));
00746
00747                               barData.Add(new double[] { underflowCount, -1.5 });
00748                               barData.Add(plot.GetFirst<Bars>().GetBaseline(new double[] { 0, -0.5 }));
00749                           }
00750                       }
00751
00752                       if (overflowCount > 0)
00753                       {
00754                           if (vertical)
00755                           {
00756                               xLabelData.Add(plot.GetFirst<Bars>().GetBaseline(new double[] { binCount +
    0.5, 0 }));
00757
00758                               barData.Add(plot.GetFirst<Bars>().GetBaseline(new double[] { binCount -
    0.5, 0 }));
00759                               barData.Add(new double[] { binCount + 0.5, overflowCount });
00760                           }
00761                           else
00762                           {
00763                               xLabelData.Add(plot.GetFirst<Bars>().GetBaseline(new double[] { 0,
    binCount + 0.5 }));
00764
00765                               barData.Add(plot.GetFirst<Bars>().GetBaseline(new double[] { 0, binCount -
```

```
      0.5 }));
00766                             barData.Add(new double[] { overflowCount, binCount + 0.5 });
00767                         }
00768                     }
00769 
00770                     Bars overUnderBars = new Bars(barData,
      plot.GetFirst<ICoordinateSystem<IReadOnlyList<double>>(), vertical) { PresentationAttributes =
      dataPresentationAttributes, GetBaseline = plot.GetFirst<Bars>().GetBaseline };
00771 
00772                     plot.AddPlotElement(overUnderBars);
00773 
00774                     Func<int, IReadOnlyList<double>, object> prevLabel = xLabels.Label;
00775                     Func<int, IReadOnlyList<double>, Point> prevMargin = xLabels.Margin;
00776 
00777                     if (vertical)
00778                     {
00779                         xLabels.Label = (i, x) =>
00780                         {
00781                             if (x[0] == -1.5)
00782                             {
00783                                 return "(-∞, " +
      underflow.ToString(System.Globalization.CultureInfo.InvariantCulture) + "]";
00784                             }
00785                             else if (x[0] == binCount + 0.5)
00786                             {
00787                                 return "[" +
      overflow.ToString(System.Globalization.CultureInfo.InvariantCulture) + ", +∞)";
00788                             }
00789                             else
00790                             {
00791                                 return prevLabel(i, x);
00792                             }
00793                         };
00794 
00795                         xLabels.Margin = (i, x) =>
00796                         {
00797                             if (x[0] == -1.5 || x[0] == binCount + 0.5)
00798                             {
00799                                 return new Point(0, 20);
00800                             }
00801                             else
00802                             {
00803                                 return prevMargin(i, x);
00804                             }
00805                         };
00806                     }
00807                     else
00808                     {
00809                         xLabels.Label = (i, x) =>
00810                         {
00811                             if (x[1] == -1.5)
00812                             {
00813                                 return "(-∞, " +
      underflow.ToString(System.Globalization.CultureInfo.InvariantCulture) + "]";
00814                             }
00815                             else if (x[1] == binCount + 0.5)
00816                             {
00817                                 return "[" +
      overflow.ToString(System.Globalization.CultureInfo.InvariantCulture) + ", +∞)";
00818                             }
00819                             else
00820                             {
00821                                 return prevLabel(i, x);
00822                             }
00823                         };
00824 
00825                         xLabels.Margin = (i, x) =>
00826                         {
00827                             if (x[1] == -1.5 || x[1] == binCount + 0.5)
00828                             {
00829                                 return new Point(-20, 0);
00830                             }
00831                             else
00832                             {
00833                                 return prevMargin(i, x);
00834                             }
00835                         };
00836 
00837                     }
00838 
00839                     xLabels.Data = xLabelData;
00840 
00841                     double ground = 0;
00842 
00843                     if (vertical)
00844                     {
00845                         ground = plot.GetFirst<ContinuousAxisTicks>().StartPoint[1];
```

```
00846                     }
00847                     else
00848                     {
00849                         ground = plot.GetAll<ContinuousAxisTicks>().ElementAt(1).StartPoint[0];
00850                     }
00851 
00852                     double[] tickStart = null;
00853                     double[] tickEnd = null;
00854 
00855                     if (overflowCount > 0 && underflowCount > 0)
00856                     {
00857                         if (vertical)
00858                         {
00859                             tickStart = new double[] { -1.5, ground };
00860                             tickEnd = new double[] { binCount + 0.5, ground };
00861                         }
00862                         else
00863                         {
00864                             tickStart = new double[] { ground, -1.5 };
00865                             tickEnd = new double[] { ground, binCount + 0.5 };
00866                         }
00867                     }
00868                     else if (overflowCount > 0)
00869                     {
00870                         if (vertical)
00871                         {
00872                             tickStart = new double[] { binCount + 0.5, ground };
00873                             tickEnd = new double[] { binCount + 0.50000001, ground };
00874                         }
00875                         else
00876                         {
00877                             tickStart = new double[] { ground, binCount + 0.5 };
00878                             tickEnd = new double[] { ground, binCount + 0.50000001 };
00879                         }
00880                     }
00881                     else if (underflowCount > 0)
00882                     {
00883                         if (vertical)
00884                         {
00885                             tickStart = new double[] { -1.5, ground };
00886                             tickEnd = new double[] { -1.50000001, ground };
00887                         }
00888                         else
00889                         {
00890                             tickStart = new double[] { ground, -1.5 };
00891                             tickEnd = new double[] { ground, -1.50000001 };
00892                         }
00893                     }
00894 
00895                     ContinuousAxisTicks overUnderTicks = new ContinuousAxisTicks(tickStart, tickEnd,
      plot.GetFirst<IContinuousCoordinateSystem>()) { IntervalCount = 1, SizeAbove = i => 3, SizeBelow = i
      => 3 };
00896 
00897                     plot.AddPlotElement(overUnderTicks);
00898                 }
00899 
00900                 return plot;
00901             }
00902 
00903             private static Plot StackedHistogram(IReadOnlyList<IReadOnlyList<double» data, bool
      vertical = true, double margin = 0.25, double width = 350, double height = 250,
00904             PlotElementPresentationAttributes axisPresentationAttributes = null,
00905             double axisArrowSize = 3,
00906             PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00907             PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00908             string xAxisTitle = null,
00909             string yAxisTitle = null,
00910             string title = null,
00911             PlotElementPresentationAttributes titlePresentationAttributes = null,
00912             PlotElementPresentationAttributes gridPresentationAttributes = null,
00913             IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
00914             IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00915             {
00916                 if (axisPresentationAttributes == null)
00917                 {
00918                     axisPresentationAttributes = new PlotElementPresentationAttributes();
00919                 }
00920 
00921                 if (gridPresentationAttributes == null)
00922                 {
00923                     gridPresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      new SolidColourBrush(Colour.FromRgb(220, 220, 220)) };
00924                 }
00925 
00926                 if (axisLabelPresentationAttributes == null)
00927                 {
00928                     axisLabelPresentationAttributes = new PlotElementPresentationAttributes() { Stroke
```

```
= null };
00929                 }
00930
00931                 if (axisTitlePresentationAttributes == null)
00932                 {
00933                     axisTitlePresentationAttributes = new PlotElementPresentationAttributes() { Font =
new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 14), Stroke =
null };
00934                 }
00935
00936                 if (dataPresentationAttributes == null)
00937                 {
00938                     dataPresentationAttributes = new PlotElementPresentationAttributes[] { new
PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(0,
114, 178)) },
00939                                                                                            new
PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(213,
94, 0)) },
00940                                                                                            new
PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(204,
121, 167)) },
00941                                                                                            new
PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(230,
159, 0)) },
00942                                                                                            new
PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(86,
180, 233)) },
00943                                                                                            new
PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(0,
158, 115)) },
00944                                                                                            new
PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(240,
228, 66)) } };
00945                 }
00946
00947                 double baselineValue = 0;
00948
00949                 if (coordinateSystem is LogarithmicCoordinateSystem2D || (!vertical &&
coordinateSystem is LinLogCoordinateSystem2D) || (vertical && coordinateSystem is
LogLinCoordinateSystem2D))
00950                 {
00951                     baselineValue = 1;
00952                 }
00953
00954                 StackedBars bars;
00955
00956                 if (baselineValue == 0)
00957                 {
00958                     bars = new StackedBars(data, coordinateSystem, vertical) { PresentationAttributes
= dataPresentationAttributes, Margin = margin };
00959                 }
00960                 else
00961                 {
00962                     bars = new StackedBars(data, new Comparison<IReadOnlyList<double>(vertical ?  new
Func<IReadOnlyList<double>, IReadOnlyList<double>, int>((x, y) => Math.Sign(x[0] - y[0])) :  (x, y) =>
Math.Sign(x[1] - y[1])), vertical ?  new Func<IReadOnlyList<double>, IReadOnlyList<double>(pt => new
double[] { pt[0], baselineValue }) :  pt => new double[] { baselineValue, pt[1] }, coordinateSystem) {
PresentationAttributes = dataPresentationAttributes, Margin = margin, Vertical = vertical };
00963                 }
00964
00965                 double[][] actualData = new double[data.Count][];
00966
00967                 for (int i = 0; i < data.Count; i++)
00968                 {
00969                     if (vertical)
00970                     {
00971                         double sum = 0;
00972
00973                         for (int j = 1; j < data[i].Count; j++)
00974                         {
00975                             sum += data[i][j];
00976                         }
00977
00978                         actualData[i] = new double[] { data[i][0], sum };
00979                     }
00980                     else
00981                     {
00982                         double sum = data[i][0];
00983
00984                         for (int j = 2; j < data[i].Count; j++)
00985                         {
00986                             sum += data[i][j];
00987                         }
00988
00989                         actualData[i] = new double[] { sum, data[i][1] };
00990                     }
00991                 }
```

```
00992
00993
00994                 (double minX, double minY, double maxX, double maxY, double rangeX, double rangeY) =
      GetDataRange(actualData);
00995
00996                 double dataMinX = minX;
00997                 double dataMinY = minY;
00998                 double dataMaxX = maxX;
00999                 double dataMaxY = maxY;
01000
01001                 if (coordinateSystem == null)
01002                 {
01003                     LinearCoordinateSystem2D linCoords = new LinearCoordinateSystem2D(data, width,
      height);
01004
01005                     if (vertical)
01006                     {
01007                         minY = Math.Min(minY, 0);
01008                         maxY = Math.Max(maxY, 0);
01009                         rangeY = maxY - minY;
01010
01011                         linCoords.MinY = minY - rangeY * 0.1;
01012                         linCoords.MaxY = maxY + rangeY * 0.1;
01013
01014
01015                         if (bars.Data.Count >= 2)
01016                         {
01017                             IReadOnlyList<double> item0 = bars.Data.ElementAt(0);
01018                             IReadOnlyList<double> item1 = bars.Data.ElementAt(1);
01019
01020                             IReadOnlyList<double> itemN = bars.Data.ElementAt(bars.Data.Count - 1);
01021                             IReadOnlyList<double> itemN1 = bars.Data.ElementAt(bars.Data.Count - 2);
01022
01023                             minX = Math.Min(minX, 1.5 * item0[0] - item1[0] * 0.5);
01024                             maxX = Math.Max(maxX, 1.5 * itemN[0] - itemN1[0] * 0.5);
01025
01026                             rangeX = maxX - minX;
01027                             linCoords.MinX = minX;
01028                             linCoords.MaxX = maxX;
01029                         }
01030                         else if (bars.Data.Count == 1)
01031                         {
01032                             minX = Math.Min(minX, -0.5);
01033                             maxX = Math.Max(maxX, 0.5);
01034                             dataMinX = minX;
01035                             dataMaxX = maxX;
01036
01037                             rangeX = maxX - minX;
01038                             linCoords.MinX = minX;
01039                             linCoords.MaxX = maxX;
01040                         }
01041                     }
01042                     else
01043                     {
01044                         minX = Math.Min(minX, 0);
01045                         maxX = Math.Max(maxX, 0);
01046                         rangeX = maxX - minX;
01047
01048                         linCoords.MinX = minX - rangeX * 0.1;
01049                         linCoords.MaxX = maxX + rangeX * 0.1;
01050
01051                         if (bars.Data.Count >= 2)
01052                         {
01053                             IReadOnlyList<double> item0 = bars.Data.ElementAt(0);
01054                             IReadOnlyList<double> item1 = bars.Data.ElementAt(1);
01055
01056                             IReadOnlyList<double> itemN = bars.Data.ElementAt(bars.Data.Count - 1);
01057                             IReadOnlyList<double> itemN1 = bars.Data.ElementAt(bars.Data.Count - 2);
01058
01059                             minY = Math.Min(minY, 1.5 * item0[1] - item1[1] * 0.5);
01060                             maxY = Math.Max(maxY, 1.5 * itemN[1] - itemN1[1] * 0.5);
01061
01062                             rangeY = maxY - minY;
01063                             linCoords.MinY = minY;
01064                             linCoords.MaxY = maxY;
01065                         }
01066                         else if (bars.Data.Count == 1)
01067                         {
01068                             minY = Math.Min(minY, -0.5);
01069                             maxY = Math.Max(maxY, 0.5);
01070                             dataMinY = minY;
01071                             dataMaxY = maxY;
01072
01073                             rangeY = maxY - minY;
01074                             linCoords.MinY = minY;
01075                             linCoords.MaxY = maxY;
01076                         }
```

```
01077                     }
01078
01079                     coordinateSystem = linCoords;
01080                     bars.CoordinateSystem = coordinateSystem;
01081                 }
01082                 else
01083                 {
01084                     if (vertical)
01085                     {
01086                         minY = Math.Min(minY, baselineValue);
01087                         rangeY = maxY - minY;
01088
01089                         if (bars.Data.Count >= 2)
01090                         {
01091                             IReadOnlyList<double> item0 = bars.Data.ElementAt(0);
01092                             IReadOnlyList<double> item1 = bars.Data.ElementAt(1);
01093
01094                             IReadOnlyList<double> itemN = bars.Data.ElementAt(bars.Data.Count - 1);
01095                             IReadOnlyList<double> itemN1 = bars.Data.ElementAt(bars.Data.Count - 2);
01096
01097                             minX = Math.Min(minX, 1.5 * item0[0] - item1[0] * 0.5);
01098                             maxX = Math.Max(maxX, 1.5 * itemN[0] - itemN1[0] * 0.5);
01099
01100                             rangeX = maxX - minX;
01101                         }
01102                         else if (bars.Data.Count == 1)
01103                         {
01104                             minX = Math.Min(minX, -0.5);
01105                             maxX = Math.Max(maxX, 0.5);
01106                             dataMinX = minX;
01107                             dataMaxX = maxX;
01108
01109                             rangeX = maxX - minX;
01110                         }
01111                     }
01112                     else
01113                     {
01114                         minX = Math.Min(minX, baselineValue);
01115                         rangeX = maxX - minX;
01116
01117                         if (bars.Data.Count >= 2)
01118                         {
01119                             IReadOnlyList<double> item0 = bars.Data.ElementAt(0);
01120                             IReadOnlyList<double> item1 = bars.Data.ElementAt(1);
01121
01122                             IReadOnlyList<double> itemN = bars.Data.ElementAt(bars.Data.Count - 1);
01123                             IReadOnlyList<double> itemN1 = bars.Data.ElementAt(bars.Data.Count - 2);
01124
01125                             minY = Math.Min(minY, 1.5 * item0[1] - item1[1] * 0.5);
01126                             maxY = Math.Max(maxY, 1.5 * itemN[1] - itemN1[1] * 0.5);
01127
01128                             rangeY = maxY - minY;
01129                         }
01130                         else if (bars.Data.Count == 1)
01131                         {
01132                             minY = Math.Min(minY, -0.5);
01133                             maxY = Math.Max(maxY, 0.5);
01134                             dataMinY = minY;
01135                             dataMaxY = maxY;
01136
01137                             rangeY = maxY - minY;
01138                         }
01139                     }
01140                 }
01141
01142
01143
01144                 if (titlePresentationAttributes == null)
01145                 {
01146                     titlePresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 18)
      };
01147                 }
01148
01149                 Point topLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, maxY });
01150                 Point topRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, maxY });
01151                 Point bottomRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, minY });
01152                 Point bottomLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, minY });
01153
01154                 double[] marginTopLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
01155                 double[] marginTopRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
01156                 double[] marginBottomRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
```

```
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
01157                 double[] marginBottomLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
01158
01159                 double[] p1 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
01160                 double[] p2 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
01161                 double[] p3 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
01162                 double[] p4 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
01163
01164
01165                 double[] p5 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
01166                 double[] p6 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
01167                 double[] p7 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
01168                 double[] p8 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
01169
01170                 if (vertical)
01171                 {
01172                     Point p1p = new Point(Math.Min(Math.Min(topLeft.X, topRight.X),
      Math.Min(bottomLeft.X, bottomRight.X)), Math.Min(Math.Min(topLeft.Y, topRight.Y),
      Math.Min(bottomLeft.Y, bottomRight.Y)) - 10);
01173                     Point p1p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMinX, dataMaxY
      });
01174
01175                     Point p2p = new Point(Math.Max(Math.Max(topLeft.X, topRight.X),
      Math.Max(bottomLeft.X, bottomRight.X)), Math.Min(Math.Min(topLeft.Y, topRight.Y),
      Math.Min(bottomLeft.Y, bottomRight.Y)) - 10);
01176                     Point p2p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMaxX, dataMaxY
      });
01177
01178                     Point p3p = new Point(Math.Min(Math.Min(topLeft.X, topRight.X),
      Math.Min(bottomLeft.X, bottomRight.X)), Math.Max(Math.Max(topLeft.Y, topRight.Y),
      Math.Max(bottomLeft.Y, bottomRight.Y)) + 10);
01179                     Point p3p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMinX, 0 });
01180
01181                     Point p4p = new Point(Math.Max(Math.Max(topLeft.X, topRight.X),
      Math.Max(bottomLeft.X, bottomRight.X)), Math.Max(Math.Max(topLeft.Y, topRight.Y),
      Math.Max(bottomLeft.Y, bottomRight.Y)) + 10);
01182                     Point p4p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMaxX, 0 });
01183
01184                     p1 = coordinateSystem.ToDataCoordinates(new Point(p1p2.X, p1p.Y));
01185                     p2 = coordinateSystem.ToDataCoordinates(new Point(p2p2.X, p2p.Y));
01186                     p3 = coordinateSystem.ToDataCoordinates(new Point(p3p2.X, p3p.Y));
01187                     p4 = coordinateSystem.ToDataCoordinates(new Point(p4p2.X, p4p.Y));
01188                 }
01189                 else
01190                 {
01191                     Point p5p = new Point(Math.Min(Math.Min(topLeft.X, topRight.X),
      Math.Min(bottomLeft.X, bottomRight.X)) - 10, Math.Min(Math.Min(topLeft.Y, topRight.Y),
      Math.Min(bottomLeft.Y, bottomRight.Y)));
01192                     Point p5p2 = coordinateSystem.ToPlotCoordinates(new double[] { 0, dataMaxY });
01193
01194                     Point p6p = new Point(Math.Min(Math.Min(topLeft.X, topRight.X),
      Math.Min(bottomLeft.X, bottomRight.X)) - 10, Math.Max(Math.Max(topLeft.Y, topRight.Y),
      Math.Max(bottomLeft.Y, bottomRight.Y)));
01195                     Point p6p2 = coordinateSystem.ToPlotCoordinates(new double[] { 0, dataMinY });
01196
01197                     Point p7p = new Point(Math.Max(Math.Max(topLeft.X, topRight.X),
      Math.Max(bottomLeft.X, bottomRight.X)) + 10, Math.Min(Math.Min(topLeft.Y, topRight.Y),
      Math.Min(bottomLeft.Y, bottomRight.Y)));
01198                     Point p7p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMaxX, dataMaxY
      });
01199
01200                     Point p8p = new Point(Math.Max(Math.Max(topLeft.X, topRight.X),
      Math.Max(bottomLeft.X, bottomRight.X)) + 10, Math.Max(Math.Max(topLeft.Y, topRight.Y),
      Math.Max(bottomLeft.Y, bottomRight.Y)));
01201                     Point p8p2 = coordinateSystem.ToPlotCoordinates(new double[] { dataMinX, dataMinY
      });
01202
01203                     p5 = coordinateSystem.ToDataCoordinates(new Point(p5p.X, p5p2.Y));
01204                     p6 = coordinateSystem.ToDataCoordinates(new Point(p6p.X, p6p2.Y));
```

```
01205                   p7 = coordinateSystem.ToDataCoordinates(new Point(p7p2.X, p7p.Y));
01206                   p8 = coordinateSystem.ToDataCoordinates(new Point(p8p2.X, p8p.Y));
01207               }
01208
01209               Grid grid;
01210
01211               if (vertical)
01212               {
01213                   grid = new Grid(p5, p6, p7, p8, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
01214               }
01215               else
01216               {
01217                   grid = new Grid(p1, p2, p3, p4, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
01218               }
01219
01220               ContinuousAxis xAxis = new ContinuousAxis(marginBottomLeft[0] < marginBottomRight[0] ?
      marginBottomLeft :  marginBottomRight, marginBottomLeft[0] < marginBottomRight[0] ?  marginBottomRight
      :  vertical ?  marginBottomLeft :
      coordinateSystem.ToDataCoordinates(coordinateSystem.ToPlotCoordinates(marginBottomLeft) + new
      Point(-axisArrowSize - 7, 0)), coordinateSystem) { PresentationAttributes =
      axisPresentationAttributes, ArrowSize = axisArrowSize };
01221               ContinuousAxis yAxis = new ContinuousAxis(marginBottomLeft[1] < marginTopLeft[1] ?
      marginBottomLeft :  marginTopLeft, marginBottomLeft[1] < marginTopLeft[1] ?  marginTopLeft :
      !vertical ?  marginBottomLeft :
      coordinateSystem.ToDataCoordinates(coordinateSystem.ToPlotCoordinates(marginBottomLeft) + new Point(0,
      axisArrowSize + 7)), coordinateSystem) { PresentationAttributes = axisPresentationAttributes,
      ArrowSize = axisArrowSize };
01222
01223               int every = (int)Math.Ceiling((double)data.Count / 5);
01224               int shift = (data.Count - every * (data.Count / every - 1) - 1) / 2;
01225
01226               ContinuousAxisTicks xTicks;
01227               ContinuousAxisTicks yTicks;
01228
01229               if (vertical)
01230               {
01231                   xTicks = new ContinuousAxisTicks(p3, p4, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes, IntervalCount = data.Count - 1, SizeAbove = i =>
      (i - shift) % every == 0 ?  3 :  2, SizeBelow = i => (i - shift) % every == 0 ?  3 :  2 };
01232                   yTicks = new ContinuousAxisTicks(p6, p5, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
01233               }
01234               else
01235               {
01236                   xTicks = new ContinuousAxisTicks(p3, p4, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
01237                   yTicks = new ContinuousAxisTicks(p6, p5, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes, IntervalCount = data.Count - 1, SizeAbove = i =>
      (i - shift) % every == 0 ?  3 :  2, SizeBelow = i => (i - shift) % every == 0 ?  3 :  2 };
01238               }
01239
01240               ContinuousAxisLabels xLabels;
01241               ContinuousAxisLabels yLabels;
01242
01243               if (vertical)
01244               {
01245                   xLabels = new ContinuousAxisLabels(p3, p4, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Alignment = TextAnchors.Center, Baseline =
      TextBaselines.Top, Rotation = 0, IntervalCount = data.Count - 1 };
01246                   yLabels = new ContinuousAxisLabels(p6, p5, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Position = _ => -10, Alignment =
      TextAnchors.Right, Rotation = 0, IntervalCount = 5 };
01247
01248                   Func<IReadOnlyList<double>, int, IEnumerable<FormattedText» originalFormatter =
      xLabels.TextFormat;
01249                   xLabels.TextFormat = (x, i) => (i - shift) % every == 0 ?  originalFormatter(x, i)
      :  null;
01250               }
01251               else
01252               {
01253                   xLabels = new ContinuousAxisLabels(p3, p4, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Alignment = TextAnchors.Center, Baseline =
      TextBaselines.Top, Rotation = 0, IntervalCount = 5 };
01254                   yLabels = new ContinuousAxisLabels(p6, p5, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Position = _ => -10, Alignment =
      TextAnchors.Right, Rotation = 0, IntervalCount = data.Count - 1 };
01255
01256                   Func<IReadOnlyList<double>, int, IEnumerable<FormattedText» originalFormatter =
      yLabels.TextFormat;
01257                   yLabels.TextFormat = (x, i) => (i - shift) % every == 0 ?  originalFormatter(x, i)
      :  null;
01258               }
01259
01260               Graphics xLabelsSize = new Graphics();
01261               xLabels.Plot(xLabelsSize);
```

```
01262                 double xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
01263
01264                 Graphics yLabelsSize = new Graphics();
01265                 yLabels.Plot(yLabelsSize);
01266                 double yLabelsWidth = yLabelsSize.GetBounds().Size.Width;
01267
01268                 ContinuousAxisTitle xTitle = new ContinuousAxisTitle(xAxisTitle, marginBottomLeft,
      marginBottomRight, coordinateSystem, axisTitlePresentationAttributes) { Position = xLabelsHeight + 20,
      Alignment = TextAnchors.Center };
01269                 ContinuousAxisTitle yTitle = new ContinuousAxisTitle(yAxisTitle, marginBottomLeft,
      marginTopLeft, coordinateSystem, axisTitlePresentationAttributes) { Position = -20 - yLabelsWidth,
      Baseline = TextBaselines.Bottom, Alignment = TextAnchors.Center };
01270
01271                 TextLabel<IReadOnlyList<double> titleLabel = new
      TextLabel<IReadOnlyList<double»(title,
      coordinateSystem.ToDataCoordinates((coordinateSystem.ToPlotCoordinates(marginTopLeft) +
      coordinateSystem.ToPlotCoordinates(marginTopRight)) * 0.5 + new Point(0, -10)), coordinateSystem) {
      Baseline = TextBaselines.Bottom, PresentationAttributes = titlePresentationAttributes };
01272
01273                 Plot tbr = new Plot();
01274                 tbr.AddPlotElements(grid, xAxis, yAxis, xTicks, yTicks, xLabels, yLabels, xTitle,
      yTitle, bars, titleLabel);
01275
01276                 return tbr;
01277             }
01278
01279 /// <summary>
01280 /// Create a new distribution plot.
01281 /// </summary>
01282 /// <param name="data">The data whose distribution will be plotted.</param>
01283 /// <param name="vertical">If this is <see langword="true"/> (the default), the distribution goes from
      the X axis up to the sampled values.  If this is <see langword="false"/>, the distribution goes from
      the Y axis to the sampled values.</param>
01284 /// <param name="binCount">The number of bins to use.  If this is &lt; 2, the number of bins is
      determined automatically using the Freedman-Diaconis rule.</param>
01285 /// <param name="smooth">If this is <see langword="false"/> (the default), the values are joined by a
      polyline.  If this is <see langword="true"/>, the values are joined by a smooth spline passing through
      all of them.</param>
01286 /// <param name="width">The width of the plot.</param>
01287 /// <param name="height">The height of the plot.</param>
01288 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
01289 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
01290 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
01291 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
01292 /// <param name="xAxisTitle">Title for the X axis.</param>
01293 /// <param name="yAxisTitle">Title for the Y axis.</param>
01294 /// <param name="title">Title for the plot.</param>
01295 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
01296 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
01297 /// <param name="dataPresentationAttributes">Presentation attributes for the plotted data.</param>
01298 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
01299 /// <returns>A <see cref="Plot"/> containing the distribution plot.</returns>
01300             public static Plot Distribution(IReadOnlyList<double> data, bool vertical = true, int
      binCount = -1, bool smooth = false, double width = 350, double height = 250,
01301                 PlotElementPresentationAttributes axisPresentationAttributes = null,
01302                 double axisArrowSize = 3,
01303                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
01304                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
01305                 string xAxisTitle = null,
01306                 string yAxisTitle = null,
01307                 string title = null,
01308                 PlotElementPresentationAttributes titlePresentationAttributes = null,
01309                 PlotElementPresentationAttributes gridPresentationAttributes = null,
01310                 PlotElementPresentationAttributes dataPresentationAttributes = null,
01311                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
01312             {
01313                 if (dataPresentationAttributes == null)
01314                 {
01315                     dataPresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      new SolidColourBrush(Colour.FromRgb(0, 114, 178)), LineWidth = 2, Fill = new
      SolidColourBrush(Colour.FromRgb(0, 114, 178).WithAlpha(0.5)) };
01316                 }
01317
01318                 double min = double.MaxValue;
01319                 double max = double.MinValue;
01320
01321                 for (int i = 0; i < data.Count; i++)
01322                 {
01323                     min = Math.Min(min, data[i]);
01324                     max = Math.Max(max, data[i]);
01325                 }
01326
01327                 if (binCount <= 1)
01328                 {
01329                     (double _, double _, double iqr) = IQR(data);
01330                     double h2 = 2 * iqr / Math.Pow(data.Count, 1.0 / 3.0);
```

```
01331
01332                     if (h2 > 0)
01333                     {
01334                         binCount = Math.Max(2, (int)Math.Ceiling((max - min) / h2));
01335                     }
01336                     else
01337                     {
01338                         binCount = 2;
01339                     }
01340                 }
01341
01342                 int[] bins = new int[binCount];
01343
01344                 if (max > min)
01345                 {
01346                     for (int i = 0; i < data.Count; i++)
01347                     {
01348                         int index = (int)Math.Min(binCount - 1, Math.Floor((data[i] - min) / (max -
      min) * binCount));
01349
01350                         bins[index]++;
01351                     }
01352                 }
01353                 else
01354                 {
01355                     bins[0] = data.Count;
01356                 }
01357
01358                 double[][] binnedData = new double[bins.Length][];
01359
01360                 for (int i = 0; i < bins.Length; i++)
01361                 {
01362                     double binStart = min + (max - min) / binCount * i;
01363                     double binEnd = min + (max - min) / binCount * (i + 1);
01364
01365                     if (vertical)
01366                     {
01367                         binnedData[i] = new double[] { (binStart + binEnd) * 0.5, bins[i] };
01368                     }
01369                     else
01370                     {
01371                         binnedData[i] = new double[] { bins[i], (binStart + binEnd) * 0.5 };
01372                     }
01373                 }
01374
01375                 Plot plot = Create.AreaChart(binnedData, vertical, smooth, width, height,
      axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, dataPresentationAttributes, coordinateSystem);
01376
01377                 return plot;
01378             }
01379
01380 /// <summary>
01381 /// Create a new stacked distribution plot.
01382 /// </summary>
01383 /// <param name="data">The data whose distribution will be plotted.</param>
01384 /// <param name="vertical">If this is <see langword="true"/> (the default), the distributions go from
      the X axis up to the sampled values.  If this is <see langword="false"/>, the distributions go from
      the Y axis to the sampled values.</param>
01385 /// <param name="binCount">The number of bins to use.  If this is &lt; 2, the number of bins is
      determined automatically using the Freedman-Diaconis rule.</param>
01386 /// <param name="smooth">If this is <see langword="false"/> (the default), the values are joined by a
      polyline.  If this is <see langword="true"/>, the values are joined by a smooth spline passing through
      all of them.</param>
01387 /// <param name="normalisationMode">The kind of normalisation to use to make the distributions
      comparable.</param>
01388 /// <param name="width">The width of the plot.</param>
01389 /// <param name="height">The height of the plot.</param>
01390 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
01391 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
01392 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
01393 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
01394 /// <param name="xAxisTitle">Title for the X axis.</param>
01395 /// <param name="yAxisTitle">Title for the Y axis.</param>
01396 /// <param name="title">Title for the plot.</param>
01397 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
01398 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
01399 /// <param name="dataPresentationAttributes">Presentation attributes for the plotted data.</param>
01400 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
01401 /// <returns>A <see cref="Plot"/> containing the stacked distribution plot.</returns>
01402             public static Plot StackedDistribution(IReadOnlyList<IReadOnlyList<double» data, bool
      vertical = true, int binCount = -1, bool smooth = false, NormalisationMode normalisationMode =
      NormalisationMode.Area, double width = 350, double height = 250,
01403                 PlotElementPresentationAttributes axisPresentationAttributes = null,
01404                 double axisArrowSize = 3,
```

```
01405                PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
01406                PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
01407                string xAxisTitle = null,
01408                string yAxisTitle = null,
01409                string title = null,
01410                PlotElementPresentationAttributes titlePresentationAttributes = null,
01411                PlotElementPresentationAttributes gridPresentationAttributes = null,
01412                IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
01413                IContinuousInvertibleCoordinateSystem coordinateSystem = null)
01414            {
01415                if (dataPresentationAttributes == null)
01416                {
01417                    dataPresentationAttributes = new PlotElementPresentationAttributes[] { new
     PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(0, 114,
     178).WithAlpha(0.5)), Stroke = new SolidColourBrush(Colour.FromRgb(0, 114, 178)) },
01418                                                                                          new
     PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(213, 94,
     0).WithAlpha(0.5)), Stroke = new SolidColourBrush(Colour.FromRgb(213, 94, 0)) },
01419                                                                                          new
     PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(204, 121,
     167).WithAlpha(0.5)), Stroke = new SolidColourBrush(Colour.FromRgb(204, 121, 167)) },
01420                                                                                          new
     PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(230, 159,
     0).WithAlpha(0.5)), Stroke = new SolidColourBrush(Colour.FromRgb(230, 159, 0)) },
01421                                                                                          new
     PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(86, 180,
     233).WithAlpha(0.5)), Stroke = new SolidColourBrush(Colour.FromRgb(86, 180, 233)) },
01422                                                                                          new
     PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(0, 158,
     115).WithAlpha(0.5)), Stroke = new SolidColourBrush(Colour.FromRgb(0, 158, 115)) },
01423                                                                                          new
     PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(240, 228,
     66).WithAlpha(0.5)), Stroke = new SolidColourBrush(Colour.FromRgb(240, 228, 66)) } };
01424                }
01425
01426                bool wasCoordinateSystemNull = coordinateSystem == null;
01427
01428                double[][][] allBinnedData = new double[data.Count][][];
01429
01430                double overallMinX = double.MaxValue;
01431                double overallMaxX = double.MinValue;
01432
01433                double overallMaxY = double.MinValue;
01434
01435                double minX0 = double.MaxValue;
01436                double maxX0 = double.MinValue;
01437                double maxY0 = double.MaxValue;
01438
01439                for (int j = 0; j < data.Count; j++)
01440                {
01441
01442                    double min = double.MaxValue;
01443                    double max = double.MinValue;
01444
01445                    for (int i = 0; i < data[j].Count; i++)
01446                    {
01447                        min = Math.Min(min, data[j][i]);
01448                        max = Math.Max(max, data[j][i]);
01449                    }
01450
01451                    if (binCount <= 1)
01452                    {
01453                        (double _, double _, double iqr) = IQR(data[j]);
01454                        double h2 = 2 * iqr / Math.Pow(data[j].Count, 1.0 / 3.0);
01455
01456                        if (h2 > 0)
01457                        {
01458                            binCount = Math.Max(2, (int)Math.Ceiling((max - min) / h2));
01459                        }
01460                        else
01461                        {
01462                            binCount = 2;
01463                        }
01464                    }
01465
01466                    int[] bins = new int[binCount];
01467
01468                    if (max > min)
01469                    {
01470                        for (int i = 0; i < data[j].Count; i++)
01471                        {
01472                            int index = (int)Math.Min(binCount - 1, Math.Floor((data[j][i] - min) /
     (max - min) * binCount));
01473
01474                            bins[index]++;
01475                        }
01476                    }
```

```
01477                     else
01478                     {
01479                         bins[0] = data[j].Count;
01480                     }
01481 
01482                     double[][] binnedData = new double[bins.Length][];
01483 
01484                     double maxBin = 1;
01485 
01486                     if (normalisationMode == NormalisationMode.Maximum)
01487                     {
01488                         maxBin = bins.Max();
01489                     }
01490                     else if (normalisationMode == NormalisationMode.None)
01491                     {
01492                         maxBin = 1;
01493                     }
01494                     else if (normalisationMode == NormalisationMode.Area)
01495                     {
01496                         maxBin = (max - min) / binCount * data[j].Count;
01497                     }
01498 
01499                     for (int i = 0; i < bins.Length; i++)
01500                     {
01501                         double binStart = min + (max - min) / binCount * i;
01502                         double binEnd = min + (max - min) / binCount * (i + 1);
01503 
01504                         if (vertical)
01505                         {
01506                             binnedData[i] = new double[] { (binStart + binEnd) * 0.5, bins[i] / maxBin
      };
01507                         }
01508                         else
01509                         {
01510                             binnedData[i] = new double[] { bins[i] / maxBin, (binStart + binEnd) * 0.5
      };
01511                         }
01512 
01513                         overallMaxY = Math.Max(overallMaxY, bins[i] / maxBin);
01514                         overallMaxX = Math.Max(overallMaxX, (binStart + binEnd) * 0.5);
01515                         overallMinX = Math.Min(overallMinX, (binStart + binEnd) * 0.5);
01516                     }
01517 
01518                     if (j == 0)
01519                     {
01520                         maxX0 = overallMaxX;
01521                         minX0 = overallMinX;
01522                         maxY0 = overallMaxY;
01523                     }
01524 
01525                     allBinnedData[j] = binnedData;
01526                 }
01527 
01528                 List<double[]> joinedBinnedData = new List<double[]>((from el in allBinnedData select
      el.Length).Sum());
01529 
01530                 for (int i = 0; i < allBinnedData.Length; i++)
01531                 {
01532                     joinedBinnedData.AddRange(allBinnedData[i]);
01533                 }
01534 
01535                 Plot plot = Create.AreaChart(joinedBinnedData, vertical, smooth, width, height,
      axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, dataPresentationAttributes[0], coordinateSystem);
01536 
01537                 Func<IReadOnlyList<double>, IReadOnlyList<double» getBaseline =
      plot.GetFirst<Area<IReadOnlyList<double»>().GetBaseline;
01538 
01539                 plot.RemovePlotElement(plot.GetFirst<Area<IReadOnlyList<double»>());
01540 
01541                 ICoordinateSystem<IReadOnlyList<double» currCoordinateSystem =
      plot.GetFirst<ICoordinateSystem<IReadOnlyList<double»>();
01542 
01543                 for (int i = 0; i < allBinnedData.Length; i++)
01544                 {
01545                     Area<IReadOnlyList<double» area = new Area<IReadOnlyList<double»(allBinnedData[i],
      getBaseline, currCoordinateSystem) { Smooth = smooth, PresentationAttributes =
      dataPresentationAttributes[i % dataPresentationAttributes.Count] };
01546                     plot.AddPlotElement(area);
01547                 }
01548 
01549                 return plot;
01550             }
01551         }
01552     }
01553 }
```

8.50 Plot.Lines.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021
00022 namespace VectSharp.Plots
00023 {
00024     partial class Plot
00025     {
00026         partial class Create
00027         {
00028 /// <summary>
00029 /// Create a new line chart containing multiple lines.
00030 /// </summary>
00031 /// <param name="data">The data to plot.</param>
00032 /// <param name="smooth">If this is <see langword="false"/> (the default), the sampled values are
      joined by a polyline.  If this is <see langword="true"/>, they are joined by a smooth spline passing
      through them.</param>
00033 /// <param name="width">The width of the plot.</param>
00034 /// <param name="height">The height of the plot.</param>
00035 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00036 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00037 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00038 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00039 /// <param name="xAxisTitle">Title for the X axis.</param>
00040 /// <param name="yAxisTitle">Title for the Y axis.</param>
00041 /// <param name="title">Title for the plot.</param>
00042 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00043 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00044 /// <param name="linePresentationAttributes">Presentation attributes for the line.</param>
00045 /// <param name="pointPresentationAttributes">Presentation attributes for the sampled points.</param>
00046 /// <param name="pointSizes">Size of the symbols drawn at the sampled points.</param>
00047 /// <param name="dataPointElements">Symbols drawn at the sampled points.</param>
00048 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00049 /// <returns>A <see cref="Plot"/> containing the line chart.</returns>
00050             public static Plot LineCharts(IReadOnlyList<IReadOnlyList<IReadOnlyList<double>> data,
      bool smooth = false, double width = 350, double height = 250,
00051                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00052                 double axisArrowSize = 3,
00053                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00054                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00055                 string xAxisTitle = null,
00056                 string yAxisTitle = null,
00057                 string title = null,
00058                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00059                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00060                 IReadOnlyList<PlotElementPresentationAttributes> linePresentationAttributes = null,
00061                 IReadOnlyList<PlotElementPresentationAttributes> pointPresentationAttributes = null,
00062                 IReadOnlyList<double> pointSizes = null,
00063                 IReadOnlyList<IDataPointElement> dataPointElements = null,
00064                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00065             {
00066                 if (axisPresentationAttributes == null)
00067                 {
00068                     axisPresentationAttributes = new PlotElementPresentationAttributes();
00069                 }
00070
00071                 if (gridPresentationAttributes == null)
00072                 {
00073                     gridPresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      new SolidColourBrush(Colour.FromRgb(220, 220, 220)) };
00074                 }
00075
00076                 if (axisLabelPresentationAttributes == null)
00077                 {
00078                     axisLabelPresentationAttributes = new PlotElementPresentationAttributes() { Stroke
      = null };
00079                 }
```

```
00080
00081                 if (axisTitlePresentationAttributes == null)
00082                 {
00083                     axisTitlePresentationAttributes = new PlotElementPresentationAttributes() { Font =
      new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 14), Stroke =
      null };
00084                 }
00085
00086                 if (linePresentationAttributes == null)
00087                 {
00088                     linePresentationAttributes = new PlotElementPresentationAttributes[] { new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(0, 114, 178)), Fill
      = null },
00089                                                                                            new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(213, 94, 0)), Fill
      = null },
00090                                                                                            new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(204, 121, 167)),
      Fill = null },
00091                                                                                            new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(230, 159, 0)), Fill
      = null },
00092                                                                                            new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(86, 180, 233)),
      Fill = null },
00093                                                                                            new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(0, 158, 115)), Fill
      = null },
00094                                                                                            new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(240, 228, 66)),
      Fill = null } };
00095                 }
00096
00097                 if (pointPresentationAttributes == null)
00098                 {
00099                     pointPresentationAttributes = (from el in linePresentationAttributes select new
      PlotElementPresentationAttributes() { Stroke = null, Fill = el.Stroke }).ToList();
00100                 }
00101
00102                 if (dataPointElements == null)
00103                 {
00104                     dataPointElements = new PathDataPointElement[] { new PathDataPointElement() };
00105                 }
00106
00107                 List<IReadOnlyList<double» allData = new List<IReadOnlyList<double»();
00108
00109                 for (int i = 0; i < data.Count; i++)
00110                 {
00111                     allData.AddRange(data[i]);
00112                 }
00113
00114                 if (coordinateSystem == null)
00115                 {
00116                     coordinateSystem = new LinearCoordinateSystem2D(allData, width, height);
00117                 }
00118
00119                 if (titlePresentationAttributes == null)
00120                 {
00121                     titlePresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 18)
      };
00122                 }
00123
00124                 (double minX, double minY, double maxX, double maxY, double rangeX, double rangeY) =
      GetDataRange(allData);
00125
00126                 Point topLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, maxY });
00127                 Point topRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, maxY });
00128                 Point bottomRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, minY });
00129                 Point bottomLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, minY });
00130
00131                 double[] marginTopLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00132                 double[] marginTopRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00133                 double[] marginBottomRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00134                 double[] marginBottomLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00135
00136                 double[] p1 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
```

```
00137                 double[] p2 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00138                 double[] p3 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00139                 double[] p4 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00140
00141
00142                 double[] p5 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00143                 double[] p6 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00144                 double[] p7 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00145                 double[] p8 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00146
00147                 Grid xGrid = new Grid(p1, p2, p3, p4, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00148                 Grid yGrid = new Grid(p5, p6, p7, p8, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00149
00150                 ContinuousAxis xAxis = new ContinuousAxis(marginBottomLeft[0] < marginBottomRight[0] ?
      marginBottomLeft :  marginBottomRight, marginBottomLeft[0] < marginBottomRight[0] ?  marginBottomRight
      :  coordinateSystem.ToDataCoordinates(coordinateSystem.ToPlotCoordinates(marginBottomLeft) + new
      Point(-axisArrowSize - 7, 0)), coordinateSystem) { PresentationAttributes =
      axisPresentationAttributes, ArrowSize = axisArrowSize };
00151                 ContinuousAxis yAxis = new ContinuousAxis(marginBottomLeft[1] < marginTopLeft[1] ?
      marginBottomLeft :  marginTopLeft, marginBottomLeft[1] < marginTopLeft[1] ?  marginTopLeft :
      coordinateSystem.ToDataCoordinates(coordinateSystem.ToPlotCoordinates(marginBottomLeft) + new Point(0,
      axisArrowSize + 7)), coordinateSystem) { PresentationAttributes = axisPresentationAttributes,
      ArrowSize = axisArrowSize };
00152
00153                 ContinuousAxisTicks xTicks = new ContinuousAxisTicks(p3, p4, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00154                 ContinuousAxisTicks yTicks = new ContinuousAxisTicks(p6, p5, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00155
00156                 ContinuousAxisLabels xLabels = new ContinuousAxisLabels(p3, p4, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Alignment = TextAnchors.Center, Baseline =
      TextBaselines.Top, Rotation = 0, IntervalCount = 5 };
00157                 ContinuousAxisLabels yLabels = new ContinuousAxisLabels(p6, p5, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Position = _ => -10, Alignment =
      TextAnchors.Right, Rotation = 0, IntervalCount = 5 };
00158
00159                 Graphics xLabelsSize = new Graphics();
00160                 xLabels.Plot(xLabelsSize);
00161                 double xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
00162
00163                 Graphics yLabelsSize = new Graphics();
00164                 yLabels.Plot(yLabelsSize);
00165                 double yLabelsWidth = yLabelsSize.GetBounds().Size.Width;
00166
00167                 ContinuousAxisTitle xTitle = new ContinuousAxisTitle(xAxisTitle, marginBottomLeft,
      marginBottomRight, coordinateSystem, axisTitlePresentationAttributes) { Position = xLabelsHeight + 20,
      Alignment = TextAnchors.Center };
00168                 ContinuousAxisTitle yTitle = new ContinuousAxisTitle(yAxisTitle, marginBottomLeft,
      marginTopLeft, coordinateSystem, axisTitlePresentationAttributes) { Position = -20 - yLabelsWidth,
      Baseline = TextBaselines.Bottom, Alignment = TextAnchors.Center };
00169
00170                 List<IPlotElement> otherElements = new List<IPlotElement>();
00171
00172                 for (int i = 0; i < data.Count; i++)
00173                 {
00174                     DataLine<IReadOnlyList<double» line1 = new DataLine<IReadOnlyList<double»(data[i],
      coordinateSystem) { PresentationAttributes = linePresentationAttributes[i %
      linePresentationAttributes.Count], Smooth = smooth };
00175
00176                     otherElements.Add(line1);
00177
00178                     if (pointSizes != null && pointSizes.Count > 0 && pointSizes[i % pointSizes.Count]
      > 0)
00179                     {
00180                         ScatterPoints<IReadOnlyList<double» points1 = new
      ScatterPoints<IReadOnlyList<double»(data[i], coordinateSystem) { PresentationAttributes =
      pointPresentationAttributes[i % pointPresentationAttributes.Count], DataPointElement =
      dataPointElements[i % dataPointElements.Count], Size = pointSizes[i % pointSizes.Count] };
00181                         otherElements.Add(points1);
00182                     }
00183
```

```
00184                 }
00185
00186                 TextLabel<IReadOnlyList<double» titleLabel = new
      TextLabel<IReadOnlyList<double»(title,
      coordinateSystem.ToDataCoordinates((coordinateSystem.ToPlotCoordinates(marginTopLeft) +
      coordinateSystem.ToPlotCoordinates(marginTopRight)) * 0.5 + new Point(0, -10)), coordinateSystem) {
      Baseline = TextBaselines.Bottom, PresentationAttributes = titlePresentationAttributes };
00187
00188                 Plot tbr = new Plot();
00189                 tbr.AddPlotElements(xGrid, yGrid, xAxis, yAxis, xTicks, yTicks, xLabels, yLabels,
      xTitle, yTitle);
00190
00191                 tbr.AddPlotElements(otherElements);
00192
00193                 tbr.AddPlotElement(titleLabel);
00194
00195                 return tbr;
00196             }
00197
00198 /// <summary>
00199 /// Create a new line chart.
00200 /// </summary>
00201 /// <param name="data">The data to plot.</param>
00202 /// <param name="smooth">If this is <see langword="false"/> (the default), the sampled values are
      joined by a polyline.  If this is <see langword="true"/>, they are joined by a smooth spline passing
      through them.</param>
00203 /// <param name="width">The width of the plot.</param>
00204 /// <param name="height">The height of the plot.</param>
00205 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00206 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00207 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00208 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00209 /// <param name="xAxisTitle">Title for the X axis.</param>
00210 /// <param name="yAxisTitle">Title for the Y axis.</param>
00211 /// <param name="title">Title for the plot.</param>
00212 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00213 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00214 /// <param name="linePresentationAttributes">Presentation attributes for the line.</param>
00215 /// <param name="pointPresentationAttributes">Presentation attributes for the sampled points.</param>
00216 /// <param name="pointSize">Size of the symbols drawn at the sampled points.</param>
00217 /// <param name="dataPointElement">Symbol drawn at the sampled points.</param>
00218 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00219 /// <returns>A <see cref="Plot"/> containing the line chart.</returns>
00220             public static Plot LineChart(IReadOnlyList<IReadOnlyList<double» data, bool smooth =
      false, double width = 350, double height = 250,
00221                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00222                 double axisArrowSize = 3,
00223                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00224                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00225                 string xAxisTitle = null,
00226                 string yAxisTitle = null,
00227                 string title = null,
00228                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00229                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00230                 PlotElementPresentationAttributes linePresentationAttributes = null,
00231                 PlotElementPresentationAttributes pointPresentationAttributes = null,
00232                 double pointSize = 0,
00233                 IDataPointElement dataPointElement = null,
00234                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00235             {
00236                 return LineCharts(new IReadOnlyList<IReadOnlyList<double»[] { data }, smooth, width,
      height, axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, linePresentationAttributes == null ?  null :  new
      PlotElementPresentationAttributes[] { linePresentationAttributes }, pointPresentationAttributes ==
      null ?  null :  new PlotElementPresentationAttributes[] { pointPresentationAttributes }, new double[]
      { pointSize }, dataPointElement == null ?  null :  new IDataPointElement[] { dataPointElement },
      coordinateSystem);
00237             }
00238
00239 /// <summary>
00240 /// Create a new line chart.
00241 /// </summary>
00242 /// <param name="data">The data to plot.</param>
00243 /// <param name="smooth">If this is <see langword="false"/> (the default), the sampled values are
      joined by a polyline.  If this is <see langword="true"/>, they are joined by a smooth spline passing
      through them.</param>
00244 /// <param name="width">The width of the plot.</param>
00245 /// <param name="height">The height of the plot.</param>
00246 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00247 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00248 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00249 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00250 /// <param name="xAxisTitle">Title for the X axis.</param>
00251 /// <param name="yAxisTitle">Title for the Y axis.</param>
00252 /// <param name="title">Title for the plot.</param>
```

```
00253 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00254 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00255 /// <param name="linePresentationAttributes">Presentation attributes for the line.</param>
00256 /// <param name="pointPresentationAttributes">Presentation attributes for the sampled points.</param>
00257 /// <param name="pointSize">Size of the symbols drawn at the sampled points.</param>
00258 /// <param name="dataPointElement">Symbol drawn at the sampled points.</param>
00259 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00260 /// <returns>A <see cref="Plot"/> containing the line chart.</returns>
00261             public static Plot LineChart(IReadOnlyList<(double, double)> data, bool smooth = false,
      double width = 350, double height = 250,
00262                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00263                 double axisArrowSize = 3,
00264                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00265                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00266                 string xAxisTitle = null,
00267                 string yAxisTitle = null,
00268                 string title = null,
00269                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00270                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00271                 PlotElementPresentationAttributes linePresentationAttributes = null,
00272                 PlotElementPresentationAttributes pointPresentationAttributes = null,
00273                 double pointSize = 0,
00274                 IDataPointElement dataPointElement = null,
00275                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00276             {
00277                 return LineCharts(new IReadOnlyList<IReadOnlyList<double>>[] { (from el in data select
      new double[] { el.Item1, el.Item2 }).ToArray() }, smooth, width, height, axisPresentationAttributes,
      axisArrowSize, axisLabelPresentationAttributes, axisTitlePresentationAttributes, xAxisTitle,
      yAxisTitle, title, titlePresentationAttributes, gridPresentationAttributes, linePresentationAttributes
      == null ?  null :  new PlotElementPresentationAttributes[] { linePresentationAttributes },
      pointPresentationAttributes == null ?  null :  new PlotElementPresentationAttributes[] {
      pointPresentationAttributes }, new double[] { pointSize }, dataPointElement == null ?  null :  new
      IDataPointElement[] { dataPointElement }, coordinateSystem);
00278             }
00279
00280 /// <summary>
00281 /// Create a new line chart.
00282 /// </summary>
00283 /// <param name="data">The data to plot.</param>
00284 /// <param name="smooth">If this is <see langword="false"/> (the default), the sampled values are
      joined by a polyline.  If this is <see langword="true"/>, they are joined by a smooth spline passing
      through them.</param>
00285 /// <param name="width">The width of the plot.</param>
00286 /// <param name="height">The height of the plot.</param>
00287 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00288 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00289 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00290 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00291 /// <param name="xAxisTitle">Title for the X axis.</param>
00292 /// <param name="yAxisTitle">Title for the Y axis.</param>
00293 /// <param name="title">Title for the plot.</param>
00294 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00295 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00296 /// <param name="linePresentationAttributes">Presentation attributes for the line.</param>
00297 /// <param name="pointPresentationAttributes">Presentation attributes for the sampled points.</param>
00298 /// <param name="pointSize">Size of the symbols drawn at the sampled points.</param>
00299 /// <param name="dataPointElement">Symbol drawn at the sampled points.</param>
00300 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00301 /// <returns>A <see cref="Plot"/> containing the line chart.</returns>
00302             public static Plot LineChart(IReadOnlyList<double> data, bool smooth = false, double width
      = 350, double height = 250,
00303                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00304                 double axisArrowSize = 3,
00305                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00306                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00307                 string xAxisTitle = null,
00308                 string yAxisTitle = null,
00309                 string title = null,
00310                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00311                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00312                 PlotElementPresentationAttributes linePresentationAttributes = null,
00313                 PlotElementPresentationAttributes pointPresentationAttributes = null,
00314                 double pointSize = 0,
00315                 IDataPointElement dataPointElement = null,
00316                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00317             {
00318                 double[][] actualData = new double[data.Count][];
00319
00320                 for (int i = 0; i < data.Count; i++)
00321                 {
00322                     actualData[i] = new double[] { i + 1, data[i] };
00323                 }
00324
00325                 Plot plot = LineChart(actualData, smooth, width, height, axisPresentationAttributes,
      axisArrowSize, axisLabelPresentationAttributes, axisTitlePresentationAttributes, xAxisTitle,
```

```
      yAxisTitle, title, titlePresentationAttributes, gridPresentationAttributes,
      linePresentationAttributes, pointPresentationAttributes, pointSize, dataPointElement,
      coordinateSystem);
00326
00327                 Plot tbr = new Plot();
00328
00329                 for (int i = 0; i < plot.PlotElements.Count; i++)
00330                 {
00331                     if (i != 2 && i != 4 && i != 6)
00332                     {
00333                         tbr.AddPlotElement(plot.PlotElements[i]);
00334                     }
00335                 }
00336
00337                 return tbr;
00338             }
00339
00340 /// <summary>
00341 /// Create a new line chart containing two lines.
00342 /// </summary>
00343 /// <param name="data1">The data to plot for the first line.</param>
00344 /// <param name="data2">The data to plot for the second line.</param>
00345 /// <param name="smooth">If this is <see langword="false"/> (the default), the sampled values are
      joined by a polyline.  If this is <see langword="true"/>, they are joined by a smooth spline passing
      through them.</param>
00346 /// <param name="width">The width of the plot.</param>
00347 /// <param name="height">The height of the plot.</param>
00348 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00349 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00350 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00351 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00352 /// <param name="xAxisTitle">Title for the X axis.</param>
00353 /// <param name="yAxisTitle">Title for the Y axis.</param>
00354 /// <param name="title">Title for the plot.</param>
00355 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00356 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00357 /// <param name="line1PresentationAttributes">Presentation attributes for the first line.</param>
00358 /// <param name="point1PresentationAttributes">Presentation attributes for the sampled points for the
      first line.</param>
00359 /// <param name="point1Size">Size of the symbols drawn at the sampled points for the first
      line.</param>
00360 /// <param name="line2PresentationAttributes">Presentation attributes for the second line.</param>
00361 /// <param name="point2PresentationAttributes">Presentation attributes for the sampled points for the
      second line.</param>
00362 /// <param name="point2Size">Size of the symbols drawn at the sampled points for the second
      line.</param>
00363 /// <param name="dataPointElement1">Symbol drawn at the sampled points for the first line.</param>
00364 /// <param name="dataPointElement2">Symbol drawn at the sampled points for the second line.</param>
00365 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00366 /// <returns>A <see cref="Plot"/> containing the line chart.</returns>
00367             public static Plot LineCharts(IReadOnlyList<IReadOnlyList<double» data1,
      IReadOnlyList<IReadOnlyList<double» data2, bool smooth = false, double width = 350, double height =
      250,
00368                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00369                 double axisArrowSize = 3,
00370                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00371                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00372                 string xAxisTitle = null,
00373                 string yAxisTitle = null,
00374                 string title = null,
00375                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00376                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00377                 PlotElementPresentationAttributes line1PresentationAttributes = null,
00378                 PlotElementPresentationAttributes point1PresentationAttributes = null,
00379                 double point1Size = 0,
00380                 PlotElementPresentationAttributes line2PresentationAttributes = null,
00381                 PlotElementPresentationAttributes point2PresentationAttributes = null,
00382                 double point2Size = 0,
00383                 IDataPointElement dataPointElement1 = null,
00384                 IDataPointElement dataPointElement2 = null,
00385                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00386             {
00387                 if (line1PresentationAttributes == null)
00388                 {
00389                     line1PresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      new SolidColourBrush(Colour.FromRgb(0, 114, 178)), Fill = null };
00390                 }
00391
00392                 if (point1PresentationAttributes == null)
00393                 {
00394                     point1PresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Fill = line1PresentationAttributes.Stroke };
00395                 }
00396
00397                 if (line2PresentationAttributes == null)
00398                 {
```

```
00399                   line2PresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      new SolidColourBrush(Colour.FromRgb(213, 94, 0)), Fill = null };
00400               }
00401
00402               if (point2PresentationAttributes == null)
00403               {
00404                   point2PresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Fill = line2PresentationAttributes.Stroke };
00405               }
00406
00407               if (dataPointElement1 == null)
00408               {
00409                   dataPointElement1 = new PathDataPointElement();
00410               }
00411
00412               if (dataPointElement2 == null)
00413               {
00414                   dataPointElement2 = new PathDataPointElement();
00415               }
00416
00417               return LineCharts(new IReadOnlyList<IReadOnlyList<double>[] { data1, data2 }, smooth,
      width, height, axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, new PlotElementPresentationAttributes[] { line1PresentationAttributes,
      line2PresentationAttributes }, new PlotElementPresentationAttributes[] { point1PresentationAttributes,
      point2PresentationAttributes }, new double[] { point1Size, point2Size }, new IDataPointElement[] {
      dataPointElement1, dataPointElement2 }, coordinateSystem);
00418           }
00419
00420 /// <summary>
00421 /// Create a new line chart containing two lines.
00422 /// </summary>
00423 /// <param name="data1">The data to plot for the first line.</param>
00424 /// <param name="data2">The data to plot for the second line.</param>
00425 /// <param name="smooth">If this is <see langword="false"/> (the default), the sampled values are
      joined by a polyline.  If this is <see langword="true"/>, they are joined by a smooth spline passing
      through them.</param>
00426 /// <param name="width">The width of the plot.</param>
00427 /// <param name="height">The height of the plot.</param>
00428 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00429 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00430 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00431 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00432 /// <param name="xAxisTitle">Title for the X axis.</param>
00433 /// <param name="yAxisTitle">Title for the Y axis.</param>
00434 /// <param name="title">Title for the plot.</param>
00435 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00436 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00437 /// <param name="line1PresentationAttributes">Presentation attributes for the first line.</param>
00438 /// <param name="point1PresentationAttributes">Presentation attributes for the sampled points for the
      first line.</param>
00439 /// <param name="point1Size">Size of the symbols drawn at the sampled points for the first
      line.</param>
00440 /// <param name="line2PresentationAttributes">Presentation attributes for the second line.</param>
00441 /// <param name="point2PresentationAttributes">Presentation attributes for the sampled points for the
      second line.</param>
00442 /// <param name="point2Size">Size of the symbols drawn at the sampled points for the second
      line.</param>
00443 /// <param name="dataPointElement1">Symbol drawn at the sampled points for the first line.</param>
00444 /// <param name="dataPointElement2">Symbol drawn at the sampled points for the second line.</param>
00445 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00446 /// <returns>A <see cref="Plot"/> containing the line chart.</returns>
00447           public static Plot LineCharts(IReadOnlyList<double> data1, IReadOnlyList<double> data2,
      bool smooth = false, double width = 350, double height = 250,
00448                   PlotElementPresentationAttributes axisPresentationAttributes = null,
00449                   double axisArrowSize = 3,
00450                   PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00451                   PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00452                   string xAxisTitle = null,
00453                   string yAxisTitle = null,
00454                   string title = null,
00455                   PlotElementPresentationAttributes titlePresentationAttributes = null,
00456                   PlotElementPresentationAttributes gridPresentationAttributes = null,
00457                   PlotElementPresentationAttributes line1PresentationAttributes = null,
00458                   PlotElementPresentationAttributes point1PresentationAttributes = null,
00459                   double point1Size = 0,
00460                   PlotElementPresentationAttributes line2PresentationAttributes = null,
00461                   PlotElementPresentationAttributes point2PresentationAttributes = null,
00462                   double point2Size = 0,
00463                   IDataPointElement dataPointElement1 = null,
00464                   IDataPointElement dataPointElement2 = null,
00465                   IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00466           {
00467               double[][] actualData1 = new double[data1.Count][];
00468
00469               for (int i = 0; i < data1.Count; i++)
```

```
00470                 {
00471                     actualData1[i] = new double[] { i + 1, data1[i] };
00472                 }
00473 
00474                 double[][] actualData2 = new double[data2.Count][];
00475 
00476                 for (int i = 0; i < data2.Count; i++)
00477                 {
00478                     actualData2[i] = new double[] { i + 1, data2[i] };
00479                 }
00480 
00481                 Plot plot = LineCharts(actualData1, actualData2, smooth, width, height,
      axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, linelPresentationAttributes, point1PresentationAttributes, point1Size,
      line2PresentationAttributes, point2PresentationAttributes, point2Size, dataPointElement1,
      dataPointElement2, coordinateSystem);
00482 
00483                 Plot tbr = new Plot();
00484 
00485                 for (int i = 0; i < plot.PlotElements.Count; i++)
00486                 {
00487                     if (i != 2 && i != 4 && i != 6)
00488                     {
00489                         tbr.AddPlotElement(plot.PlotElements[i]);
00490                     }
00491                 }
00492 
00493                 return tbr;
00494             }
00495 
00496 /// <summary>
00497 /// Create a new line chart containing multiple lines.
00498 /// </summary>
00499 /// <param name="data">The data to plot.</param>
00500 /// <param name="smooth">If this is <see langword="false"/> (the default), the sampled values are
      joined by a polyline.  If this is <see langword="true"/>, they are joined by a smooth spline passing
      through them.</param>
00501 /// <param name="width">The width of the plot.</param>
00502 /// <param name="height">The height of the plot.</param>
00503 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00504 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00505 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00506 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00507 /// <param name="xAxisTitle">Title for the X axis.</param>
00508 /// <param name="yAxisTitle">Title for the Y axis.</param>
00509 /// <param name="title">Title for the plot.</param>
00510 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00511 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00512 /// <param name="linePresentationAttributes">Presentation attributes for the line.</param>
00513 /// <param name="pointPresentationAttributes">Presentation attributes for the sampled points.</param>
00514 /// <param name="pointSizes">Size of the symbols drawn at the sampled points.</param>
00515 /// <param name="dataPointElements">Symbols drawn at the sampled points.</param>
00516 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00517 /// <returns>A <see cref="Plot"/> containing the line chart.</returns>
00518             public static Plot LineCharts(IReadOnlyList<IReadOnlyList<(double, double)» data, bool
      smooth = false, double width = 350, double height = 250,
00519                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00520                 double axisArrowSize = 3,
00521                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00522                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00523                 string xAxisTitle = null,
00524                 string yAxisTitle = null,
00525                 string title = null,
00526                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00527                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00528                 IReadOnlyList<PlotElementPresentationAttributes> linePresentationAttributes = null,
00529                 IReadOnlyList<PlotElementPresentationAttributes> pointPresentationAttributes = null,
00530                 IReadOnlyList<double> pointSizes = null,
00531                 IReadOnlyList<IDataPointElement> dataPointElements = null,
00532                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00533             {
00534                 return LineCharts((from el in data select (from el2 in el select new double[] {
      el2.Item1, el2.Item2 }).ToArray()).ToArray(), smooth, width, height, axisPresentationAttributes,
      axisArrowSize, axisLabelPresentationAttributes, axisTitlePresentationAttributes, xAxisTitle,
      yAxisTitle, title, titlePresentationAttributes, gridPresentationAttributes,
      linePresentationAttributes, pointPresentationAttributes, pointSizes, dataPointElements,
      coordinateSystem);
00535             }
00536 
00537 /// <summary>
00538 /// Create a new line chart containing multiple lines.
00539 /// </summary>
00540 /// <param name="data">The data to plot.</param>
00541 /// <param name="smooth">If this is <see langword="false"/> (the default), the sampled values are
      joined by a polyline.  If this is <see langword="true"/>, they are joined by a smooth spline passing
```

```
through them.</param>
00542 /// <param name="width">The width of the plot.</param>
00543 /// <param name="height">The height of the plot.</param>
00544 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00545 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00546 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00547 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00548 /// <param name="xAxisTitle">Title for the X axis.</param>
00549 /// <param name="yAxisTitle">Title for the Y axis.</param>
00550 /// <param name="title">Title for the plot.</param>
00551 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00552 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00553 /// <param name="linePresentationAttributes">Presentation attributes for the line.</param>
00554 /// <param name="pointPresentationAttributes">Presentation attributes for the sampled points.</param>
00555 /// <param name="pointSizes">Size of the symbols drawn at the sampled points.</param>
00556 /// <param name="dataPointElements">Symbols drawn at the sampled points.</param>
00557 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
to plot space.</param>
00558 /// <returns>A <see cref="Plot"/> containing the line chart.</returns>
00559             public static Plot LineCharts(IReadOnlyList<IReadOnlyList<double» data, bool smooth =
false, double width = 350, double height = 250,
00560                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00561                 double axisArrowSize = 3,
00562                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00563                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00564                 string xAxisTitle = null,
00565                 string yAxisTitle = null,
00566                 string title = null,
00567                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00568                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00569                 IReadOnlyList<PlotElementPresentationAttributes> linePresentationAttributes = null,
00570                 IReadOnlyList<PlotElementPresentationAttributes> pointPresentationAttributes = null,
00571                 IReadOnlyList<double> pointSizes = null,
00572                 IReadOnlyList<IDataPointElement> dataPointElements = null,
00573                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00574             {
00575                 double[][][] actualData = new double[data.Count][][];
00576 
00577                 for (int i = 0; i < data.Count; i++)
00578                 {
00579                     actualData[i] = new double[data[i].Count][];
00580 
00581                     for (int j = 0; j < data[i].Count; j++)
00582                     {
00583                         actualData[i][j] = new double[] { j + 1, data[i][j] };
00584                     }
00585                 }
00586 
00587 
00588                 Plot plot = LineCharts(actualData, smooth, width, height, axisPresentationAttributes,
axisArrowSize, axisLabelPresentationAttributes, axisTitlePresentationAttributes, xAxisTitle,
yAxisTitle, title, titlePresentationAttributes, gridPresentationAttributes,
linePresentationAttributes, pointPresentationAttributes, pointSizes, dataPointElements,
coordinateSystem);
00589 
00590                 Plot tbr = new Plot();
00591 
00592                 for (int i = 0; i < plot.PlotElements.Count; i++)
00593                 {
00594                     if (i != 2 && i != 4 && i != 6)
00595                     {
00596                         tbr.AddPlotElement(plot.PlotElements[i]);
00597                     }
00598                 }
00599 
00600                 return tbr;
00601             }
00602         }
00603     }
00604 }
```

8.51 Plot.Pie.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021
00022 namespace VectSharp.Plots
00023 {
00024     partial class Plot
00025     {
00026         partial class Create
00027         {
00028 /// <summary>
00029 /// Create a new doughnut chart containing multiple doughnuts.
00030 /// </summary>
00031 /// <param name="data">The data to plot.</param>
00032 /// <param name="clockwise">If this is <see langword="false"/> (the default), the slices are drawn in
      anti-clockwise direction.  If this is <see langword="true"/>, they are drawn in clockwise
      direction.</param>
00033 /// <param name="startAngle">The initial angle at which the slices are drawn.</param>
00034 /// <param name="innerRadius">The radius of the doughnut "hole".</param>
00035 /// <param name="width">The width of the plot.</param>
00036 /// <param name="height">The height of the plot.</param>
00037 /// <param name="title">Title for the plot.</param>
00038 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00039 /// <param name="dataPresentationAttributes">Presentation attributes for the slices.</param>
00040 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00041 /// <returns>A <see cref="Plot"/> containing the doughnut chart.</returns>
00042             public static Plot DoughnutCharts(IReadOnlyList<IReadOnlyList<double» data, bool clockwise
      = false, double startAngle = 0, double innerRadius = 0.5, double width = 350, double height = 250,
00043                 string title = null,
00044                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00045                 IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
00046                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00047             {
00048
00049                 if (dataPresentationAttributes == null)
00050                 {
00051                     dataPresentationAttributes = new PlotElementPresentationAttributes[] { new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(0,
      114, 178)) },
00052                                                                                            new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(213,
      94, 0)) },
00053                                                                                            new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(204,
      121, 167)) },
00054                                                                                            new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(230,
      159, 0)) },
00055                                                                                            new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(86,
      180, 233)) },
00056                                                                                            new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(0,
      158, 115)) },
00057                                                                                            new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(240,
      228, 66)) } };
00058                 }
00059
00060                 int rowCount = (int)Math.Round(Math.Sqrt(data.Count));
00061
00062                 int[] elementsPerRow = new int[rowCount];
00063
00064                 for (int i = 0; i < rowCount; i++)
00065                 {
00066                     elementsPerRow[i] = data.Count / rowCount;
00067
00068                     if (data.Count - (data.Count / rowCount) * rowCount > i)
00069                     {
00070                         elementsPerRow[i]++;
00071                     }
00072                 }
00073
00074                 if (coordinateSystem == null)
00075                 {
00076                     double pieWidth = 2.5 * elementsPerRow.Max() + 2;
00077                     double pieHeight = 2.5 * rowCount + 2;
00078
00079                     double aspectRatio = pieWidth / pieHeight;
```

```
00080                     double targetAspectRatio = width / height;
00081
00082                     if (targetAspectRatio > aspectRatio)
00083                     {
00084                         pieWidth = targetAspectRatio * pieHeight;
00085                     }
00086                     else
00087                     {
00088                         pieHeight = pieWidth / targetAspectRatio;
00089                     }
00090
00091                     coordinateSystem = new LinearCoordinateSystem2D(-1, pieWidth - 1, -pieHeight + 1,
      1, width, height);
00092                 }
00093
00094                 if (titlePresentationAttributes == null)
00095                 {
00096                     titlePresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 18)
      };
00097                 }
00098
00099                 Plot tbr = new Plot();
00100
00101
00102                 for (int i = 0; i < data.Count; i++)
00103                 {
00104                     int column = i;
00105                     int row = 0;
00106
00107                     for (int j = 0; j < rowCount; j++)
00108                     {
00109                         if (elementsPerRow[j] <= column)
00110                         {
00111                             column -= elementsPerRow[j];
00112                             row++;
00113                         }
00114                         else
00115                         {
00116                             break;
00117                         }
00118                     }
00119
00120                     Pie pie = new Pie(data[i], new double[] { column * 2.5 + 1.25 *
      (elementsPerRow.Max() - elementsPerRow[row]), -row * 2.5 }, new double[] { 1, 1 }, coordinateSystem)
00121                     {
00122                         PresentationAttributes = dataPresentationAttributes,
00123                         Clockwise = clockwise,
00124                         StartAngle = startAngle,
00125                         InnerRadius = new double[] { innerRadius, innerRadius }
00126                     };
00127
00128                     tbr.AddPlotElement(pie);
00129                 }
00130
00131                 Point top = coordinateSystem.ToPlotCoordinates(new double[] { (elementsPerRow.Max() -
      1) * 0.5 * 2.5, 1 });
00132                 double[] titleCentre = coordinateSystem.ToDataCoordinates(new Point(top.X, top.Y -
      10));
00133
00134                 TextLabel<IReadOnlyList<double» titleLabel = new
      TextLabel<IReadOnlyList<double»(title, titleCentre, coordinateSystem) { Baseline =
      TextBaselines.Bottom, PresentationAttributes = titlePresentationAttributes };
00135                 tbr.AddPlotElement(titleLabel);
00136
00137
00138                 return tbr;
00139             }
00140
00141 /// <summary>
00142 /// Create a new doughnut chart.
00143 /// </summary>
00144 /// <param name="data">The data to plot.</param>
00145 /// <param name="clockwise">If this is <see langword="false"/> (the default), the slices are drawn in
      anti-clockwise direction.  If this is <see langword="true"/>, they are drawn in clockwise
      direction.</param>
00146 /// <param name="startAngle">The initial angle at which the slices are drawn.</param>
00147 /// <param name="innerRadius">The radius of the doughnut "hole".</param>
00148 /// <param name="width">The width of the plot.</param>
00149 /// <param name="height">The height of the plot.</param>
00150 /// <param name="title">Title for the plot.</param>
00151 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00152 /// <param name="dataPresentationAttributes">Presentation attributes for the slices.</param>
00153 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00154 /// <returns>A <see cref="Plot"/> containing the doughnut chart.</returns>
00155             public static Plot DoughnutChart(IReadOnlyList<double> data, bool clockwise = false,
```

```
    double startAngle = 0, double innerRadius = 0.5, double width = 350, double height = 250,
00156              string title = null,
00157              PlotElementPresentationAttributes titlePresentationAttributes = null,
00158              IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
00159              IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00160             {
00161                 return DoughnutCharts(new IReadOnlyList<double>[] { data }, clockwise, startAngle,
    innerRadius, width, height, title, titlePresentationAttributes, dataPresentationAttributes,
    coordinateSystem);
00162             }
00163
00164
00165 /// <summary>
00166 /// Create a new pie chart.
00167 /// </summary>
00168 /// <param name="data">The data to plot.</param>
00169 /// <param name="clockwise">If this is <see langword="false"/> (the default), the slices are drawn in
    anti-clockwise direction.  If this is <see langword="true"/>, they are drawn in clockwise
    direction.</param>
00170 /// <param name="startAngle">The initial angle at which the slices are drawn.</param>
00171 /// <param name="width">The width of the plot.</param>
00172 /// <param name="height">The height of the plot.</param>
00173 /// <param name="title">Title for the plot.</param>
00174 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00175 /// <param name="dataPresentationAttributes">Presentation attributes for the slices.</param>
00176 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
    to plot space.</param>
00177 /// <returns>A <see cref="Plot"/> containing the pie chart.</returns>
00178             public static Plot PieChart(IReadOnlyList<double> data, bool clockwise = false, double
    startAngle = 0, double width = 350, double height = 250,
00179              string title = null,
00180              PlotElementPresentationAttributes titlePresentationAttributes = null,
00181              IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
00182              IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00183             {
00184                 return DoughnutCharts(new IReadOnlyList<double>[] { data }, clockwise, startAngle, 0,
    width, height, title, titlePresentationAttributes, dataPresentationAttributes, coordinateSystem);
00185             }
00186
00187 /// <summary>
00188 /// Create a new pie chart containing multiple pies.
00189 /// </summary>
00190 /// <param name="data">The data to plot.</param>
00191 /// <param name="clockwise">If this is <see langword="false"/> (the default), the slices are drawn in
    anti-clockwise direction.  If this is <see langword="true"/>, they are drawn in clockwise
    direction.</param>
00192 /// <param name="startAngle">The initial angle at which the slices are drawn.</param>
00193 /// <param name="width">The width of the plot.</param>
00194 /// <param name="height">The height of the plot.</param>
00195 /// <param name="title">Title for the plot.</param>
00196 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00197 /// <param name="dataPresentationAttributes">Presentation attributes for the slices.</param>
00198 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
    to plot space.</param>
00199 /// <returns>A <see cref="Plot"/> containing the pie chart.</returns>
00200             public static Plot PieCharts(IReadOnlyList<IReadOnlyList<double» data, bool clockwise =
    false, double startAngle = 0, double width = 350, double height = 250,
00201                 string title = null,
00202                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00203                 IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
00204                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00205             {
00206                 return DoughnutCharts(data, clockwise, startAngle, 0, width, height, title,
    titlePresentationAttributes, dataPresentationAttributes, coordinateSystem);
00207             }
00208
00209 /// <summary>
00210 /// Create a new doughnut chart containing multiple concentric doughnuts.
00211 /// </summary>
00212 /// <param name="data">The data to plot.</param>
00213 /// <param name="clockwise">If this is <see langword="false"/> (the default), the slices are drawn in
    anti-clockwise direction.  If this is <see langword="true"/>, they are drawn in clockwise
    direction.</param>
00214 /// <param name="startAngle">The initial angle at which the slices are drawn.</param>
00215 /// <param name="innerRadius">The radius of the innermost doughnut "hole".  If this is <see
    cref="double.NaN"/>, it is determined automatically based on the number of concentric
    doughnuts.</param>
00216 /// <param name="margin">The spacing between consecutive doughnut rings, ranging from 0 to 1.</param>
00217 /// <param name="width">The width of the plot.</param>
00218 /// <param name="height">The height of the plot.</param>
00219 /// <param name="title">Title for the plot.</param>
00220 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00221 /// <param name="dataPresentationAttributes">Presentation attributes for the slices.</param>
00222 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
    to plot space.</param>
00223 /// <returns>A <see cref="Plot"/> containing the doughnut chart.</returns>
00224             public static Plot StackedDoughnutChart(IReadOnlyList<IReadOnlyList<double» data, bool
```

```
      clockwise = false, double startAngle = 0, double innerRadius = double.NaN, double margin = 0.1, double
      width = 350, double height = 250,
00225                 string title = null,
00226                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00227                 IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
00228                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00229             {
00230
00231                 if (dataPresentationAttributes == null)
00232                 {
00233                     dataPresentationAttributes = new PlotElementPresentationAttributes[] { new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(0, 114, 178)), Fill
      = new SolidColourBrush(Colour.FromRgb(0, 114, 178)), LineWidth = 0.5 },
00234                                                                                                 new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(213, 94, 0)), Fill
      = new SolidColourBrush(Colour.FromRgb(213, 94, 0)), LineWidth = 0.5 },
00235                                                                                                 new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(204, 121, 167)),
      Fill = new SolidColourBrush(Colour.FromRgb(204, 121, 167)), LineWidth = 0.5 },
00236                                                                                                 new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(230, 159, 0)), Fill
      = new SolidColourBrush(Colour.FromRgb(230, 159, 0)), LineWidth = 0.5 },
00237                                                                                                 new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(86, 180, 233)),
      Fill = new SolidColourBrush(Colour.FromRgb(86, 180, 233)), LineWidth = 0.5 },
00238                                                                                                 new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(0, 158, 115)), Fill
      = new SolidColourBrush(Colour.FromRgb(0, 158, 115)), LineWidth = 0.5 },
00239                                                                                                 new
      PlotElementPresentationAttributes() { Stroke = new SolidColourBrush(Colour.FromRgb(240, 228, 66)),
      Fill = new SolidColourBrush(Colour.FromRgb(240, 228, 66)), LineWidth = 0.5 } };
00240                 }
00241
00242                 if (double.IsNaN(innerRadius))
00243                 {
00244                     innerRadius = Math.Max(0.1, Math.Pow(2, -data.Count));
00245                 }
00246
00247                 if (coordinateSystem == null)
00248                 {
00249                     double pieWidth = 2.5 + 2;
00250                     double pieHeight = 2.5 + 2;
00251
00252                     double aspectRatio = pieWidth / pieHeight;
00253                     double targetAspectRatio = width / height;
00254
00255                     if (targetAspectRatio > aspectRatio)
00256                     {
00257                         pieWidth = targetAspectRatio * pieHeight;
00258                     }
00259                     else
00260                     {
00261                         pieHeight = pieWidth / targetAspectRatio;
00262                     }
00263
00264                     coordinateSystem = new LinearCoordinateSystem2D(-1, pieWidth - 1, -pieHeight + 1,
      1, width, height);
00265                 }
00266
00267                 if (titlePresentationAttributes == null)
00268                 {
00269                     titlePresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 18)
      };
00270                 }
00271
00272                 double actualMargin = margin * (1 - innerRadius) / data.Count;
00273
00274                 Plot tbr = new Plot();
00275
00276                 for (int i = 0; i < data.Count; i++)
00277                 {
00278
00279                     Pie pie = new Pie(data[i], new double[] { 0, 0 }, new double[] { innerRadius + (1
      - innerRadius) * (i + 1) / data.Count - actualMargin * 0.5, innerRadius + (1 - innerRadius) * (i + 1)
      / data.Count - actualMargin * 0.5 }, coordinateSystem)
00280                     {
00281                         PresentationAttributes = dataPresentationAttributes,
00282                         Clockwise = clockwise,
00283                         StartAngle = startAngle,
00284                         InnerRadius = new double[] { innerRadius + (1 - innerRadius) * i / data.Count
      + actualMargin * 0.5, innerRadius + (1 - innerRadius) * i / data.Count + actualMargin * 0.5 }
00285                     };
00286
00287                     tbr.AddPlotElement(pie);
00288                 }
00289
```

```
00290                 Point top = coordinateSystem.ToPlotCoordinates(new double[] { 0, 1 });
00291                 double[] titleCentre = coordinateSystem.ToDataCoordinates(new Point(top.X, top.Y -
      10));
00292 
00293                 TextLabel<IReadOnlyList<double» titleLabel = new
      TextLabel<IReadOnlyList<double»(title, titleCentre, coordinateSystem) { Baseline =
      TextBaselines.Bottom, PresentationAttributes = titlePresentationAttributes };
00294                 tbr.AddPlotElement(titleLabel);
00295 
00296 
00297                 return tbr;
00298             }
00299         }
00300     }
00301 }
```

8.52 Plot.ScatterPlot.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021 
00022 namespace VectSharp.Plots
00023 {
00024     partial class Plot
00025     {
00026         partial class Create
00027         {
00028 /// <summary>
00029 /// Create a new scatter plot.
00030 /// </summary>
00031 /// <param name="data">The data to plot.</param>
00032 /// <param name="width">The width of the plot.</param>
00033 /// <param name="height">The height of the plot.</param>
00034 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00035 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00036 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00037 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00038 /// <param name="xAxisTitle">Title for the X axis.</param>
00039 /// <param name="yAxisTitle">Title for the Y axis.</param>
00040 /// <param name="title">Title for the plot.</param>
00041 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00042 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00043 /// <param name="dataPresentationAttributes">Presentation attributes for the data.</param>
00044 /// <param name="pointSizes">Size of the symbols drawn at the sampled points.</param>
00045 /// <param name="dataPointElements">Symbols drawn at the sampled points.</param>
00046 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00047 /// <returns>A <see cref="Plot"/> containing the scatter plot.</returns>
00048             public static Plot ScatterPlot(IReadOnlyList<IReadOnlyList<IReadOnlyList<double»> data,
      double width = 350, double height = 250,
00049                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00050                 double axisArrowSize = 3,
00051                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00052                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00053                 string xAxisTitle = null,
00054                 string yAxisTitle = null,
00055                 string title = null,
00056                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00057                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00058                 IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
00059                 IReadOnlyList<double> pointSizes = null,
00060                 IReadOnlyList<IDataPointElement> dataPointElements = null,
00061                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00062             {
00063                 if (axisPresentationAttributes == null)
```

```
00064                 {
00065                     axisPresentationAttributes = new PlotElementPresentationAttributes();
00066                 }
00067 
00068                 if (gridPresentationAttributes == null)
00069                 {
00070                     gridPresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      new SolidColourBrush(Colour.FromRgb(220, 220, 220)) };
00071                 }
00072 
00073                 if (axisLabelPresentationAttributes == null)
00074                 {
00075                     axisLabelPresentationAttributes = new PlotElementPresentationAttributes() { Stroke
      = null };
00076                 }
00077 
00078                 if (axisTitlePresentationAttributes == null)
00079                 {
00080                     axisTitlePresentationAttributes = new PlotElementPresentationAttributes() { Font =
      new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 14), Stroke =
      null };
00081                 }
00082 
00083                 if (dataPresentationAttributes == null)
00084                 {
00085                     dataPresentationAttributes = new PlotElementPresentationAttributes[] { new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(0,
      114, 178)) },
00086                                                                                            new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(213,
      94, 0)) },
00087                                                                                            new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(204,
      121, 167)) },
00088                                                                                            new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(230,
      159, 0)) },
00089                                                                                            new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(86,
      180, 233)) },
00090                                                                                            new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(0,
      158, 115)) },
00091                                                                                            new
      PlotElementPresentationAttributes() { Stroke = null, Fill = new SolidColourBrush(Colour.FromRgb(240,
      228, 66)) } };
00092                 }
00093 
00094                 if (dataPointElements == null)
00095                 {
00096                     dataPointElements = new PathDataPointElement[] { new PathDataPointElement() };
00097                 }
00098 
00099                 if (pointSizes == null)
00100                 {
00101                     pointSizes = new double[] { 2 };
00102                 }
00103 
00104                 List<IReadOnlyList<double» allData = data.Aggregate(new List<IReadOnlyList<double»(),
      (a, b) => { a.AddRange(b); return a; });
00105 
00106                 if (coordinateSystem == null)
00107                 {
00108                     coordinateSystem = new LinearCoordinateSystem2D(allData, width, height);
00109                 }
00110 
00111                 if (titlePresentationAttributes == null)
00112                 {
00113                     titlePresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 18)
      };
00114                 }
00115 
00116                 (double minX, double minY, double maxX, double maxY, double rangeX, double rangeY) =
      GetDataRange(allData);
00117 
00118                 Point topLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, maxY });
00119                 Point topRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, maxY });
00120                 Point bottomRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, minY });
00121                 Point bottomLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, minY });
00122 
00123                 double[] marginTopLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00124                 double[] marginTopRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
```

```
00125                 double[] marginBottomRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00126                 double[] marginBottomLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00127
00128                 double[] p1 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00129                 double[] p2 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00130                 double[] p3 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00131                 double[] p4 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00132
00133
00134                 double[] p5 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00135                 double[] p6 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00136                 double[] p7 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00137                 double[] p8 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00138
00139                 Grid xGrid = new Grid(p1, p2, p3, p4, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00140                 Grid yGrid = new Grid(p5, p6, p7, p8, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00141
00142                 ContinuousAxis xAxis = new ContinuousAxis(marginBottomLeft[0] < marginBottomRight[0] ?
      marginBottomLeft :  marginBottomRight, marginBottomLeft[0] < marginBottomRight[0] ?  marginBottomRight
      :  coordinateSystem.ToDataCoordinates(coordinateSystem.ToPlotCoordinates(marginBottomLeft) + new
      Point(-axisArrowSize - 7, 0)), coordinateSystem) { PresentationAttributes =
      axisPresentationAttributes, ArrowSize = axisArrowSize };
00143                 ContinuousAxis yAxis = new ContinuousAxis(marginBottomLeft[1] < marginTopLeft[1] ?
      marginBottomLeft :  marginTopLeft, marginBottomLeft[1] < marginTopLeft[1] ?  marginTopLeft :
      coordinateSystem.ToDataCoordinates(coordinateSystem.ToPlotCoordinates(marginBottomLeft) + new Point(0,
      axisArrowSize + 7)), coordinateSystem) { PresentationAttributes = axisPresentationAttributes,
      ArrowSize = axisArrowSize };
00144
00145                 ContinuousAxisTicks xTicks = new ContinuousAxisTicks(p3, p4, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00146                 ContinuousAxisTicks yTicks = new ContinuousAxisTicks(p6, p5, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00147
00148                 ContinuousAxisLabels xLabels = new ContinuousAxisLabels(p3, p4, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Alignment = TextAnchors.Center, Baseline =
      TextBaselines.Top, Rotation = 0, IntervalCount = 5 };
00149                 ContinuousAxisLabels yLabels = new ContinuousAxisLabels(p6, p5, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Position = _ => -10, Alignment =
      TextAnchors.Right, Rotation = 0, IntervalCount = 5 };
00150
00151                 Graphics xLabelsSize = new Graphics();
00152                 xLabels.Plot(xLabelsSize);
00153                 double xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
00154
00155                 Graphics yLabelsSize = new Graphics();
00156                 yLabels.Plot(yLabelsSize);
00157                 double yLabelsWidth = yLabelsSize.GetBounds().Size.Width;
00158
00159                 ContinuousAxisTitle xTitle = new ContinuousAxisTitle(xAxisTitle, marginBottomLeft,
      marginBottomRight, coordinateSystem, axisTitlePresentationAttributes) { Position = xLabelsHeight + 20,
      Alignment = TextAnchors.Center };
00160                 ContinuousAxisTitle yTitle = new ContinuousAxisTitle(yAxisTitle, marginBottomLeft,
      marginTopLeft, coordinateSystem, axisTitlePresentationAttributes) { Position = -20 - yLabelsWidth,
      Baseline = TextBaselines.Bottom, Alignment = TextAnchors.Center };
00161
00162                 Plot tbr = new Plot();
00163                 tbr.AddPlotElements(xGrid, yGrid, xAxis, yAxis, xTicks, yTicks, xLabels, yLabels,
      xTitle, yTitle);
00164
00165                 for (int i = 0; i < data.Count; i++)
00166                 {
00167                     ScatterPoints<IReadOnlyList<double» points = new
      ScatterPoints<IReadOnlyList<double»(data[i], coordinateSystem) { PresentationAttributes =
      dataPresentationAttributes[i % dataPresentationAttributes.Count], Size = pointSizes[i %
      pointSizes.Count], DataPointElement = dataPointElements[i % dataPointElements.Count] };
```

```
00168                   tbr.AddPlotElement(points);
00169               }
00170
00171
00172               TextLabel<IReadOnlyList<double» titleLabel = new
    TextLabel<IReadOnlyList<double»(title,
    coordinateSystem.ToDataCoordinates((coordinateSystem.ToPlotCoordinates(marginTopLeft) +
    coordinateSystem.ToPlotCoordinates(marginTopRight)) * 0.5 + new Point(0, -10)), coordinateSystem) {
    Baseline = TextBaselines.Bottom, PresentationAttributes = titlePresentationAttributes };
00173
00174               tbr.AddPlotElement(titleLabel);
00175
00176               return tbr;
00177           }
00178
00179 /// <summary>
00180 /// Create a new scatter plot.
00181 /// </summary>
00182 /// <param name="data">The data to plot.</param>
00183 /// <param name="width">The width of the plot.</param>
00184 /// <param name="height">The height of the plot.</param>
00185 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00186 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00187 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00188 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00189 /// <param name="xAxisTitle">Title for the X axis.</param>
00190 /// <param name="yAxisTitle">Title for the Y axis.</param>
00191 /// <param name="title">Title for the plot.</param>
00192 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00193 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00194 /// <param name="dataPresentationAttributes">Presentation attributes for the data.</param>
00195 /// <param name="pointSize">Size of the symbols drawn at the sampled points.</param>
00196 /// <param name="dataPointElement">Symbol drawn at the sampled points.</param>
00197 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
    to plot space.</param>
00198 /// <returns>A <see cref="Plot"/> containing the scatter plot.</returns>
00199           public static Plot ScatterPlot(IReadOnlyList<IReadOnlyList<double» data, double width =
    350, double height = 250,
00200               PlotElementPresentationAttributes axisPresentationAttributes = null,
00201               double axisArrowSize = 3,
00202               PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00203               PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00204               string xAxisTitle = null,
00205               string yAxisTitle = null,
00206               string title = null,
00207               PlotElementPresentationAttributes titlePresentationAttributes = null,
00208               PlotElementPresentationAttributes gridPresentationAttributes = null,
00209               PlotElementPresentationAttributes dataPresentationAttributes = null,
00210               double pointSize = 2,
00211               IDataPointElement dataPointElement = null,
00212               IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00213           {
00214               return ScatterPlot(new IReadOnlyList<IReadOnlyList<double»[] { data }, width, height,
    axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
    axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
    gridPresentationAttributes, dataPresentationAttributes == null ?  null :  new
    PlotElementPresentationAttributes[] { dataPresentationAttributes }, new double[] { pointSize },
    dataPointElement == null ?  null :  new IDataPointElement[] { dataPointElement }, coordinateSystem);
00215           }
00216
00217 /// <summary>
00218 /// Create a new scatter plot.
00219 /// </summary>
00220 /// <param name="data">The data to plot.</param>
00221 /// <param name="width">The width of the plot.</param>
00222 /// <param name="height">The height of the plot.</param>
00223 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00224 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00225 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00226 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00227 /// <param name="xAxisTitle">Title for the X axis.</param>
00228 /// <param name="yAxisTitle">Title for the Y axis.</param>
00229 /// <param name="title">Title for the plot.</param>
00230 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00231 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00232 /// <param name="dataPresentationAttributes">Presentation attributes for the data.</param>
00233 /// <param name="pointSizes">Size of the symbols drawn at the sampled points.</param>
00234 /// <param name="dataPointElements">Symbols drawn at the sampled points.</param>
00235 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
    to plot space.</param>
00236 /// <returns>A <see cref="Plot"/> containing the scatter plot.</returns>
00237           public static Plot ScatterPlot(IReadOnlyList<IReadOnlyList<(double, double)» data, double
    width = 350, double height = 250,
00238               PlotElementPresentationAttributes axisPresentationAttributes = null,
00239               double axisArrowSize = 3,
00240               PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00241               PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
```

```
00242                string xAxisTitle = null,
00243                string yAxisTitle = null,
00244                string title = null,
00245                PlotElementPresentationAttributes titlePresentationAttributes = null,
00246                PlotElementPresentationAttributes gridPresentationAttributes = null,
00247                IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
00248                IReadOnlyList<double> pointSizes = null,
00249                IReadOnlyList<IDataPointElement> dataPointElements = null,
00250                IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00251            {
00252                return ScatterPlot((from el in data select (from el2 in el select new double[] {
      el2.Item1, el2.Item2 }).ToArray()).ToArray(), width, height, axisPresentationAttributes,
      axisArrowSize, axisLabelPresentationAttributes, axisTitlePresentationAttributes, xAxisTitle,
      yAxisTitle, title, titlePresentationAttributes, gridPresentationAttributes,
      dataPresentationAttributes, pointSizes, dataPointElements, coordinateSystem);
00253            }
00254
00255 /// <summary>
00256 /// Create a new scatter plot.
00257 /// </summary>
00258 /// <param name="data">The data to plot.</param>
00259 /// <param name="width">The width of the plot.</param>
00260 /// <param name="height">The height of the plot.</param>
00261 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00262 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00263 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00264 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00265 /// <param name="xAxisTitle">Title for the X axis.</param>
00266 /// <param name="yAxisTitle">Title for the Y axis.</param>
00267 /// <param name="title">Title for the plot.</param>
00268 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00269 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00270 /// <param name="dataPresentationAttributes">Presentation attributes for the data.</param>
00271 /// <param name="pointSize">Size of the symbols drawn at the sampled points.</param>
00272 /// <param name="dataPointElement">Symbol drawn at the sampled points.</param>
00273 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00274 /// <returns>A <see cref="Plot"/> containing the scatter plot.</returns>
00275            public static Plot ScatterPlot(IReadOnlyList<(double, double)> data, double width = 350,
      double height = 250,
00276            PlotElementPresentationAttributes axisPresentationAttributes = null,
00277            double axisArrowSize = 3,
00278            PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00279            PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00280            string xAxisTitle = null,
00281            string yAxisTitle = null,
00282            string title = null,
00283            PlotElementPresentationAttributes titlePresentationAttributes = null,
00284            PlotElementPresentationAttributes gridPresentationAttributes = null,
00285            PlotElementPresentationAttributes dataPresentationAttributes = null,
00286            double pointSize = 2,
00287            IDataPointElement dataPointElement = null,
00288            IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00289            {
00290                return ScatterPlot((from el in data select new double[] { el.Item1, el.Item2
      }).ToArray(), width, height, axisPresentationAttributes, axisArrowSize,
      axisLabelPresentationAttributes, axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title,
      titlePresentationAttributes, gridPresentationAttributes, dataPresentationAttributes, pointSize,
      dataPointElement, coordinateSystem);
00291            }
00292
00293 /// <summary>
00294 /// Create a new scatter plot.
00295 /// </summary>
00296 /// <param name="data">The data to plot.</param>
00297 /// <param name="width">The width of the plot.</param>
00298 /// <param name="height">The height of the plot.</param>
00299 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00300 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00301 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00302 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00303 /// <param name="xAxisTitle">Title for the X axis.</param>
00304 /// <param name="yAxisTitle">Title for the Y axis.</param>
00305 /// <param name="title">Title for the plot.</param>
00306 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00307 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00308 /// <param name="dataPresentationAttributes">Presentation attributes for the data.</param>
00309 /// <param name="pointSize">Size of the symbols drawn at the sampled points.</param>
00310 /// <param name="dataPointElement">Symbol drawn at the sampled points.</param>
00311 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00312 /// <returns>A <see cref="Plot"/> containing the scatter plot.</returns>
00313            public static Plot ScatterPlot(IReadOnlyList<Point> data, double width = 350, double
      height = 250,
00314                PlotElementPresentationAttributes axisPresentationAttributes = null,
00315                double axisArrowSize = 3,
00316                PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
```

```
00317                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00318                 string xAxisTitle = null,
00319                 string yAxisTitle = null,
00320                 string title = null,
00321                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00322                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00323                 PlotElementPresentationAttributes dataPresentationAttributes = null,
00324                 double pointSize = 2,
00325                 IDataPointElement dataPointElement = null,
00326                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00327             {
00328                 return ScatterPlot((from el in data select new double[] { el.X, el.Y }).ToArray(),
      width, height, axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, dataPresentationAttributes, pointSize, dataPointElement,
      coordinateSystem);
00329             }
00330
00331 /// <summary>
00332 /// Create a new scatter plot.
00333 /// </summary>
00334 /// <param name="data">The data to plot.</param>
00335 /// <param name="width">The width of the plot.</param>
00336 /// <param name="height">The height of the plot.</param>
00337 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00338 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00339 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00340 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00341 /// <param name="xAxisTitle">Title for the X axis.</param>
00342 /// <param name="yAxisTitle">Title for the Y axis.</param>
00343 /// <param name="title">Title for the plot.</param>
00344 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00345 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00346 /// <param name="dataPresentationAttributes">Presentation attributes for the data.</param>
00347 /// <param name="pointSizes">Size of the symbols drawn at the sampled points.</param>
00348 /// <param name="dataPointElements">Symbols drawn at the sampled points.</param>
00349 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00350 /// <returns>A <see cref="Plot"/> containing the scatter plot.</returns>
00351             public static Plot ScatterPlot(IReadOnlyList<IReadOnlyList<Point» data, double width =
      350, double height = 250,
00352                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00353                 double axisArrowSize = 3,
00354                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00355                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00356                 string xAxisTitle = null,
00357                 string yAxisTitle = null,
00358                 string title = null,
00359                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00360                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00361                 IReadOnlyList<PlotElementPresentationAttributes> dataPresentationAttributes = null,
00362                 IReadOnlyList<double> pointSizes = null,
00363                 IReadOnlyList<IDataPointElement> dataPointElements = null,
00364                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00365             {
00366                 return ScatterPlot((from el in data select (from el2 in el select new double[] {
      el2.X, el2.Y }).ToArray()).ToArray(), width, height, axisPresentationAttributes, axisArrowSize,
      axisLabelPresentationAttributes, axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title,
      titlePresentationAttributes, gridPresentationAttributes, dataPresentationAttributes, pointSizes,
      dataPointElements, coordinateSystem);
00367             }
00368         }
00369     }
00370 }
```

8.53 Plot.SwarmPlot.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2024 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
```

```
00017
00018 using MathNet.Numerics.Statistics;
00019 using System;
00020 using System.Collections.Generic;
00021 using System.Linq;
00022
00023 namespace VectSharp.Plots
00024 {
00025     partial class Plot
00026     {
00027         partial class Create
00028         {
00029 /// <summary>
00030 /// Create a new swarm plot.
00031 /// </summary>
00032 /// <param name="data">The data to plot.</param>
00033 /// <param name="vertical">If this is <see langword="true"/> (the default), the swarms are parallel to the Y axis.  If this is <see langword="false"/>, the swarms are parallel to the X axis.</param>
00034 /// <param name="showBoxPlots">If this is <see langword="true"/> (the default), a box plot is drawn on top of each swarm plot.</param>
00035 /// <param name="boxWidth">The width of the box plots, as a proportion of the available space for each swarm.</param>
00036 /// <param name="spacing">The spacing between consecutive swarms, in data space coordinates.</param>
00037 /// <param name="dataRangeMin">If this is not <see langword="null"/>, this value is used to override the default minimum value for the plot.  Useful if you wish to create multiple plots with the same scale,
00038 /// even though the sampled values have different ranges.</param>
00039 /// <param name="dataRangeMax">If this is not <see langword="null"/>, this value is used to override the default maximum value for the plot.  Useful if you wish to create multiple plots with the same scale,
00040 /// even though the sampled values have different ranges.</param>
00041 /// <param name="width">The width of the plot.</param>
00042 /// <param name="height">The height of the plot.</param>
00043 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00044 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00045 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00046 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00047 /// <param name="xAxisTitle">Title for the X axis.</param>
00048 /// <param name="yAxisTitle">Title for the Y axis.</param>
00049 /// <param name="title">Title for the plot.</param>
00050 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00051 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00052 /// <param name="swarmSymbol">The symbol drawn at each data point.</param>
00053 /// <param name="swarmPointSize">The size of the symbol drawn at each data point.</param>
00054 /// <param name="swarmPointMargin">The margin between data point symbols.  Set this to a negative value to compress the plot, or to a positive value to expand it.</param>
00055 /// <param name="swarmPresentationAttributes">Presentation attributes for the swarm data points.</param>
00056 /// <param name="boxPresentationAttributes">Presentation attributes for the box plots.</param>
00057 /// <param name="medianPresentationAttributes">Presentation attributes for the symbol drawn at the medians.</param>
00058 /// <param name="medianSymbol">The symbol drawn at the medians.</param>
00059 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space to plot space.</param>
00060 /// <returns>A <see cref="Plot"/> containing the swarm plot.</returns>
00061             public static Plot SwarmPlot<T>(IReadOnlyList<(T, IReadOnlyList<double>)> data, bool showBoxPlots = true, double boxWidth = 0.04, double spacing = 10, double?  dataRangeMin = null, double?  dataRangeMax = null, bool vertical = true, double width = 350, double height = 250,
00062                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00063                 double axisArrowSize = 3,
00064                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00065                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00066                 string xAxisTitle = null,
00067                 string yAxisTitle = null,
00068                 string title = null,
00069                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00070                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00071                 IDataPointElement swarmSymbol = null,
00072                 double swarmPointSize = 2,
00073                 double swarmPointMargin = 0.25,
00074                 IReadOnlyList<PlotElementPresentationAttributes> swarmPresentationAttributes = null,
00075                 IReadOnlyList<PlotElementPresentationAttributes> boxPresentationAttributes = null,
00076                 IReadOnlyList<PlotElementPresentationAttributes> medianPresentationAttributes = null,
00077                 IDataPointElement medianSymbol = null,
00078                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00079             {
00080                 if (swarmPresentationAttributes == null)
00081                 {
00082                     if (showBoxPlots)
00083                     {
00084                         swarmPresentationAttributes = new PlotElementPresentationAttributes[] { new PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(197, 235, 255)), Stroke = new SolidColourBrush(Colour.FromRgb(0, 114, 178)), LineWidth = 0 },
00085                                                                                                 new PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 233, 218)), Stroke = new SolidColourBrush(Colour.FromRgb(213, 94, 0)), LineWidth = 0 },
00086                                                                                                 new
```

```
PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 222, 240)),
Stroke = new SolidColourBrush(Colour.FromRgb(204, 121, 167)), LineWidth = 0 },
00087                                                                                   new
PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 242, 216)),
Stroke = new SolidColourBrush(Colour.FromRgb(230, 159, 0)), LineWidth = 0 },
00088                                                                                   new
PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(214, 241, 255)),
Stroke = new SolidColourBrush(Colour.FromRgb(86, 180, 233)), LineWidth = 0 },
00089                                                                                   new
PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(203, 255, 239)),
Stroke = new SolidColourBrush(Colour.FromRgb(0, 158, 115)), LineWidth = 0 },
00090                                                                                   new
PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 249, 189)),
Stroke = new SolidColourBrush(Colour.FromRgb(240, 228, 66)), LineWidth = 0 } };
00091                     }
00092                     else
00093                     {
00094                         swarmPresentationAttributes = new PlotElementPresentationAttributes[] { new
PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(0, 114, 178)), Stroke
= null },
00095                                                                                   new
PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(213, 94, 0)), Stroke
= null },
00096                                                                                   new
PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(204, 121, 167)),
Stroke = null },
00097                                                                                   new
PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(230, 159, 0)), Stroke
= null },
00098                                                                                   new
PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(86, 180, 233)),
Stroke = null },
00099                                                                                   new
PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(0, 158, 115)), Stroke
= null },
00100                                                                                   new
PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(240, 228, 66)),
Stroke = null } };
00101                     }
00102                 }
00103
00104                 if (boxPresentationAttributes == null)
00105                 {
00106                     PlotElementPresentationAttributes[] temp = new
PlotElementPresentationAttributes[swarmPresentationAttributes.Count];
00107
00108                     for (int i = 0; i < swarmPresentationAttributes.Count; i++)
00109                     {
00110                         temp[i] = new PlotElementPresentationAttributes() { Stroke =
swarmPresentationAttributes[i].Stroke, Fill = swarmPresentationAttributes[i].Stroke };
00111                     }
00112
00113                     boxPresentationAttributes = temp;
00114                 }
00115
00116                 if (medianPresentationAttributes == null)
00117                 {
00118                     PlotElementPresentationAttributes[] temp = new
PlotElementPresentationAttributes[swarmPresentationAttributes.Count];
00119
00120                     for (int i = 0; i < swarmPresentationAttributes.Count; i++)
00121                     {
00122                         temp[i] = new PlotElementPresentationAttributes() { Stroke = null, Fill =
swarmPresentationAttributes[i].Fill };
00123                     }
00124
00125                     medianPresentationAttributes = temp;
00126                 }
00127
00128                 if (medianSymbol == null)
00129                 {
00130                     medianSymbol = new PathDataPointElement();
00131                 }
00132
00133                 if (swarmSymbol == null)
00134                 {
00135                     swarmSymbol = new PathDataPointElement();
00136                 }
00137
00138                 if (axisPresentationAttributes == null)
00139                 {
00140                     axisPresentationAttributes = new PlotElementPresentationAttributes();
00141                 }
00142
00143                 if (gridPresentationAttributes == null)
00144                 {
00145                     gridPresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
```

```
      new SolidColourBrush(Colour.FromRgb(220, 220, 220)) };
00146                 }
00147
00148                 if (axisLabelPresentationAttributes == null)
00149                 {
00150                     axisLabelPresentationAttributes = new PlotElementPresentationAttributes() { Stroke
      = null };
00151                 }
00152
00153                 if (axisTitlePresentationAttributes == null)
00154                 {
00155                     axisTitlePresentationAttributes = new PlotElementPresentationAttributes() { Font =
      new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 14), Stroke =
      null };
00156                 }
00157
00158                 if (titlePresentationAttributes == null)
00159                 {
00160                     titlePresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 18)
      };
00161                 }
00162
00163                 double[] medians = new double[data.Count];
00164                 double[] firstQuartiles = new double[data.Count];
00165                 double[] thirdQuartiles = new double[data.Count];
00166                 double[] upperWhiskers = new double[data.Count];
00167                 double[] lowerWhiskers = new double[data.Count];
00168
00169                 double minData = double.MaxValue;
00170                 double maxData = double.MinValue;
00171
00172                 for (int i = 0; i < data.Count; i++)
00173                 {
00174                     medians[i] = data[i].Item2.Median();
00175                     firstQuartiles[i] = data[i].Item2.LowerQuartile();
00176                     thirdQuartiles[i] = data[i].Item2.UpperQuartile();
00177
00178
00179                     upperWhiskers[i] = thirdQuartiles[i] + (thirdQuartiles[i] - firstQuartiles[i]) *
      1.5;
00180                     lowerWhiskers[i] = firstQuartiles[i] - (thirdQuartiles[i] - firstQuartiles[i]) *
      1.5;
00181
00182                     double uW = double.MinValue;
00183                     double lW = double.MaxValue;
00184
00185                     for (int j = 0; j < data[i].Item2.Count; j++)
00186                     {
00187                         if (data[i].Item2[j] <= upperWhiskers[i])
00188                         {
00189                             uW = Math.Max(uW, data[i].Item2[j]);
00190                         }
00191
00192                         if (data[i].Item2[j] >= lowerWhiskers[i])
00193                         {
00194                             lW = Math.Min(lW, data[i].Item2[j]);
00195                         }
00196
00197                         minData = Math.Min(minData, data[i].Item2[j]);
00198                         maxData = Math.Max(maxData, data[i].Item2[j]);
00199                     }
00200
00201                     upperWhiskers[i] = uW;
00202                     lowerWhiskers[i] = lW;
00203                 }
00204
00205                 double minX, maxX, minY, maxY;
00206
00207                 if (vertical)
00208                 {
00209                     minX = 0;
00210                     maxX = spacing * data.Count;
00211
00212                     minY = minData;
00213                     maxY = maxData;
00214
00215                     if (dataRangeMin != null)
00216                     {
00217                         minY = dataRangeMin.Value;
00218                     }
00219
00220                     if (dataRangeMax != null)
00221                     {
00222                         maxY = dataRangeMax.Value;
00223                     }
00224                 }
```

```
00225                 else
00226                 {
00227                     minY = 0;
00228                     maxY = spacing * data.Count;
00229 
00230                     minX = minData;
00231                     maxX = maxData;
00232 
00233                     if (dataRangeMin != null)
00234                     {
00235                         minX = dataRangeMin.Value;
00236                     }
00237 
00238                     if (dataRangeMax != null)
00239                     {
00240                         maxX = dataRangeMax.Value;
00241                     }
00242                 }
00243 
00244                 if (coordinateSystem == null)
00245                 {
00246                     coordinateSystem = new LinearCoordinateSystem2D(minX, maxX, minY, maxY, width,
      height);
00247                 }
00248 
00249                 Point topLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, maxY });
00250                 Point topRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, maxY });
00251                 Point bottomRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, minY });
00252                 Point bottomLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, minY });
00253 
00254                 double[] marginTopLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00255                 double[] marginTopRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00256                 double[] marginBottomRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00257                 double[] marginBottomLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00258 
00259                 double[] p1 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00260                 double[] p2 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00261                 double[] p3 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00262                 double[] p4 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00263 
00264 
00265                 double[] p5 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00266                 double[] p6 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00267                 double[] p7 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00268                 double[] p8 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00269 
00270                 ContinuousAxis xAxis = new ContinuousAxis(marginBottomLeft[0] < marginBottomRight[0] ?
      marginBottomLeft :  marginBottomRight, marginBottomLeft[0] < marginBottomRight[0] ?  marginBottomRight
      :  vertical ?  marginBottomLeft :
      coordinateSystem.ToDataCoordinates(coordinateSystem.ToPlotCoordinates(marginBottomLeft) + new
      Point(-axisArrowSize - 7, 0)), coordinateSystem) { PresentationAttributes =
      axisPresentationAttributes, ArrowSize = vertical ?  0 :  axisArrowSize };
00271                 ContinuousAxis yAxis = new ContinuousAxis(marginBottomLeft[1] < marginTopLeft[1] ?
      marginBottomLeft :  marginTopLeft, marginBottomLeft[1] < marginTopLeft[1] ?  marginTopLeft :
      !vertical ?  marginBottomLeft :
      coordinateSystem.ToDataCoordinates(coordinateSystem.ToPlotCoordinates(marginBottomLeft) + new Point(0,
      axisArrowSize + 7)), coordinateSystem) { PresentationAttributes = axisPresentationAttributes,
      ArrowSize = vertical ?  axisArrowSize :  0 };
00272 
00273 
00274                 IPlotElement xTicks;
00275                 IPlotElement yTicks;
00276 
```

```
00277                 IPlotElement xLabels;
00278                 IPlotElement yLabels;
00279 
00280                 if (vertical)
00281                 {
00282                     xTicks = new ScatterPoints<IReadOnlyList<double>>((from el in Enumerable.Range(0,
      data.Count) select new double[] { spacing * (el + 0.5), p3[1] }), coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes, Size = 1, DataPointElement = new
      PathDataPointElement(new GraphicsPath().MoveTo(0, -3).LineTo(0, 3)) };
00283                     yTicks = new ContinuousAxisTicks(p6, p5, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00284 
00285                     xLabels = new DataLabels<IReadOnlyList<double>>((from el in Enumerable.Range(0,
      data.Count) select new double[] { spacing * (el + 0.5), p3[1] }), coordinateSystem) { Alignment =
      TextAnchors.Center, Baseline = TextBaselines.Top, Label = (i, _) => data[i].Item1,
      PresentationAttributes = axisLabelPresentationAttributes, Margin = (a, b) => new Point(0, 10) };
00286                     yLabels = new ContinuousAxisLabels(p6, p5, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Position = _ => -10, Alignment =
      TextAnchors.Right, Rotation = 0, IntervalCount = 5 };
00287                 }
00288                 else
00289                 {
00290                     xTicks = new ContinuousAxisTicks(p3, p4, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00291 
00292                     yTicks = new ScatterPoints<IReadOnlyList<double>>((from el in Enumerable.Range(0,
      data.Count) select new double[] { p6[0], spacing * (el + 0.5) }), coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes, Size = 1, DataPointElement = new
      PathDataPointElement(new GraphicsPath().MoveTo(-3, 0).LineTo(3, 0)) };
00293 
00294                     xLabels = new ContinuousAxisLabels(p3, p4, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Alignment = TextAnchors.Center, Baseline =
      TextBaselines.Top, Rotation = 0, IntervalCount = 5 };
00295                     yLabels = new DataLabels<IReadOnlyList<double>>((from el in Enumerable.Range(0,
      data.Count) select new double[] { p6[0], spacing * (el + 0.5) }), coordinateSystem) { Alignment =
      TextAnchors.Right, Baseline = TextBaselines.Middle, Label = (i, _) => data[i].Item1,
      PresentationAttributes = axisLabelPresentationAttributes, Margin = (a, b) => new Point(-10, 0) };
00296                 }
00297 
00298                 Graphics xLabelsSize = new Graphics();
00299                 xLabels.Plot(xLabelsSize);
00300                 double xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
00301 
00302                 Graphics yLabelsSize = new Graphics();
00303                 yLabels.Plot(yLabelsSize);
00304                 double yLabelsWidth = yLabelsSize.GetBounds().Size.Width;
00305 
00306                 ContinuousAxisTitle xTitle = new ContinuousAxisTitle(xAxisTitle, marginBottomLeft,
      marginBottomRight, coordinateSystem, axisTitlePresentationAttributes) { Position = xLabelsHeight + 20,
      Alignment = TextAnchors.Center };
00307                 ContinuousAxisTitle yTitle = new ContinuousAxisTitle(yAxisTitle, marginBottomLeft,
      marginTopLeft, coordinateSystem, axisTitlePresentationAttributes) { Position = -20 - yLabelsWidth,
      Baseline = TextBaselines.Bottom, Alignment = TextAnchors.Center };
00308 
00309                 Plot plot = new Plot();
00310 
00311                 if (vertical)
00312                 {
00313                     Grid yGrid = new Grid(p5, p6, p7, p8, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00314                     plot.AddPlotElement(yGrid);
00315                 }
00316                 else
00317                 {
00318                     Grid xGrid = new Grid(p1, p2, p3, p4, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00319                     plot.AddPlotElement(xGrid);
00320                 }
00321 
00322                 plot.AddPlotElements(xAxis, yAxis, xTicks, yTicks, xLabels, yLabels, xTitle, yTitle);
00323 
00324                 double maxCount = (from el in data select el.Item2.Count).Max();
00325 
00326                 for (int i = 0; i < data.Count; i++)
00327                 {
00328                     if (vertical)
00329                     {
00330                         Swarm swarm = new Swarm(new double[] { spacing * (i + 0.5), 0 }, new double[]
      { 0, 1 }, data[i].Item2, coordinateSystem)
00331                         {
00332                             PointMargin = swarmPointMargin,
00333                             PointSize = swarmPointSize,
00334                             PresentationAttributes = swarmPresentationAttributes[i %
      swarmPresentationAttributes.Count],
00335                             SwarmPointElement = swarmSymbol
00336                         };
00337                         plot.AddPlotElement(swarm);
```

```
00338
00339                       if (showBoxPlots)
00340                       {
00341                           BoxPlot box = new BoxPlot(new double[] { spacing * (i + 0.5), medians[i]
      }, new double[] { 0, 1 }, lowerWhiskers[i] - medians[i], firstQuartiles[i] - medians[i],
      thirdQuartiles[i] - medians[i], upperWhiskers[i] - medians[i], coordinateSystem)
00342                           {
00343                               BoxPresentationAttributes = boxPresentationAttributes[i %
      boxPresentationAttributes.Count],
00344                               WhiskersPresentationAttributes = boxPresentationAttributes[i %
      boxPresentationAttributes.Count],
00345                               Width = spacing * boxWidth * 0.5,
00346                               NotchSize = 0,
00347                               WhiskerWidth = 0,
00348                               CentreSymbolPresentationAttributes = medianPresentationAttributes[i %
      medianPresentationAttributes.Count],
00349                               CentreSymbol = medianSymbol
00350                           };
00351                           plot.AddPlotElement(box);
00352                       }
00353                   }
00354                   else
00355                   {
00356                       Swarm swarm = new Swarm(new double[] { 0, spacing * (i + 0.5) }, new double[]
      { 1, 0 }, data[i].Item2, coordinateSystem)
00357                       {
00358                           PointMargin = swarmPointMargin,
00359                           PointSize = swarmPointSize,
00360                           PresentationAttributes = swarmPresentationAttributes[i %
      swarmPresentationAttributes.Count],
00361                           SwarmPointElement = swarmSymbol
00362                       };
00363                       plot.AddPlotElement(swarm);
00364
00365                       if (showBoxPlots)
00366                       {
00367                           BoxPlot box = new BoxPlot(new double[] { medians[i], spacing * (i + 0.5)
      }, new double[] { 1, 0 }, lowerWhiskers[i] - medians[i], firstQuartiles[i] - medians[i],
      thirdQuartiles[i] - medians[i], upperWhiskers[i] - medians[i], coordinateSystem)
00368                           {
00369                               BoxPresentationAttributes = boxPresentationAttributes[i %
      boxPresentationAttributes.Count],
00370                               WhiskersPresentationAttributes = boxPresentationAttributes[i %
      boxPresentationAttributes.Count],
00371                               Width = spacing * boxWidth * 0.5,
00372                               NotchSize = 0,
00373                               WhiskerWidth = 0,
00374                               CentreSymbolPresentationAttributes = medianPresentationAttributes[i %
      medianPresentationAttributes.Count],
00375                               CentreSymbol = medianSymbol
00376                           };
00377
00378                           plot.AddPlotElement(box);
00379                       }
00380                   }
00381               }
00382
00383               TextLabel<IReadOnlyList<double» titleLabel = new
      TextLabel<IReadOnlyList<double»(title,
      coordinateSystem.ToDataCoordinates((coordinateSystem.ToPlotCoordinates(marginTopLeft) +
      coordinateSystem.ToPlotCoordinates(marginTopRight)) * 0.5 + new Point(0, -10)), coordinateSystem) {
      Baseline = TextBaselines.Bottom, PresentationAttributes = titlePresentationAttributes };
00384
00385               plot.AddPlotElement(titleLabel);
00386
00387               return plot;
00388           }
00389
00390
00391 /// <summary>
00392 /// Create a new swarm plot.
00393 /// </summary>
00394 /// <param name="data">The data to plot.</param>
00395 /// <param name="vertical">If this is <see langword="true"/> (the default), the swarms are parallel to
      the Y axis.  If this is <see langword="false"/>, the swarms are parallel to the X axis.</param>
00396 /// <param name="showBoxPlots">If this is <see langword="true"/> (the default), a box plot is drawn on
      top of each swarm plot.</param>
00397 /// <param name="boxWidth">The width of the box plots, as a proportion of the available space for each
      swarm.</param>
00398 /// <param name="spacing">The spacing between consecutive swarms, in data space coordinates.</param>
00399 /// <param name="dataRangeMin">If this is not <see langword="null"/>, this value is used to override
      the default minimum value for the plot.  Useful if you wish to create multiple plots with the same
      scale,
00400 /// even though the sampled values have different ranges.</param>
00401 /// <param name="dataRangeMax">If this is not <see langword="null"/>, this value is used to override
      the default maximum value for the plot.  Useful if you wish to create multiple plots with the same
      scale,
```

```
00402 /// even though the sampled values have different ranges.</param>
00403 /// <param name="width">The width of the plot.</param>
00404 /// <param name="height">The height of the plot.</param>
00405 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00406 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00407 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00408 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00409 /// <param name="xAxisTitle">Title for the X axis.</param>
00410 /// <param name="yAxisTitle">Title for the Y axis.</param>
00411 /// <param name="title">Title for the plot.</param>
00412 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00413 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00414 /// <param name="swarmSymbol">The symbol drawn at each data point.</param>
00415 /// <param name="swarmPointSize">The size of the symbol drawn at each data point.</param>
00416 /// <param name="swarmPointMargin">The margin between data point symbols.  Set this to a negative
      value to compress the plot, or to a positive value to expand it.</param>
00417 /// <param name="swarmPresentationAttributes">Presentation attributes for the swarm data
      points.</param>
00418 /// <param name="boxPresentationAttributes">Presentation attributes for the box plots.</param>
00419 /// <param name="medianPresentationAttributes">Presentation attributes for the symbol drawn at the
      medians.</param>
00420 /// <param name="medianSymbol">The symbol drawn at the medians.</param>
00421 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00422 /// <returns>A <see cref="Plot"/> containing the swarm plot.</returns>
00423             public static Plot SwarmPlot(IReadOnlyList<IReadOnlyList<double» data, bool showBoxPlots =
      true, double boxWidth = 0.04, double spacing = 10, double?  dataRangeMin = null, double?  dataRangeMax
      = null, bool vertical = true, double width = 350, double height = 250,
00424                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00425                 double axisArrowSize = 3,
00426                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00427                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00428                 string xAxisTitle = null,
00429                 string yAxisTitle = null,
00430                 string title = null,
00431                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00432                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00433                 IDataPointElement swarmSymbol = null,
00434                 double swarmPointSize = 2,
00435                 double swarmPointMargin = 0.25,
00436                 IReadOnlyList<PlotElementPresentationAttributes> swarmPresentationAttributes = null,
00437                 IReadOnlyList<PlotElementPresentationAttributes> boxPresentationAttributes = null,
00438                 IReadOnlyList<PlotElementPresentationAttributes> medianPresentationAttributes = null,
00439                 IDataPointElement medianSymbol = null,
00440                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00441             {
00442                 return SwarmPlot((from el in data select ((string)null, el)).ToArray(), showBoxPlots,
      boxWidth, spacing, dataRangeMin, dataRangeMax, vertical, width, height, axisPresentationAttributes,
      axisArrowSize, axisLabelPresentationAttributes, axisTitlePresentationAttributes, xAxisTitle,
      yAxisTitle, title, titlePresentationAttributes, gridPresentationAttributes, swarmSymbol,
      swarmPointSize, swarmPointMargin, swarmPresentationAttributes, boxPresentationAttributes,
      medianPresentationAttributes, medianSymbol, coordinateSystem);
00443             }
00444
00445 /// <summary>
00446 /// Create a new swarm plot.
00447 /// </summary>
00448 /// <param name="data">The data to plot.</param>
00449 /// <param name="vertical">If this is <see langword="true"/> (the default), the swarms are parallel to
      the Y axis.  If this is <see langword="false"/>, the swarms are parallel to the X axis.</param>
00450 /// <param name="showBoxPlots">If this is <see langword="true"/> (the default), a box plot is drawn on
      top of each swarm plot.</param>
00451 /// <param name="boxWidth">The width of the box plots, as a proportion of the available space for each
      swarm.</param>
00452 /// <param name="spacing">The spacing between consecutive swarms, in data space coordinates.</param>
00453 /// <param name="dataRangeMin">If this is not <see langword="null"/>, this value is used to override
      the default minimum value for the plot.  Useful if you wish to create multiple plots with the same
      scale,
00454 /// even though the sampled values have different ranges.</param>
00455 /// <param name="dataRangeMax">If this is not <see langword="null"/>, this value is used to override
      the default maximum value for the plot.  Useful if you wish to create multiple plots with the same
      scale,
00456 /// even though the sampled values have different ranges.</param>
00457 /// <param name="width">The width of the plot.</param>
00458 /// <param name="height">The height of the plot.</param>
00459 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00460 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00461 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00462 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00463 /// <param name="xAxisTitle">Title for the X axis.</param>
00464 /// <param name="yAxisTitle">Title for the Y axis.</param>
00465 /// <param name="title">Title for the plot.</param>
00466 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00467 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00468 /// <param name="swarmSymbol">The symbol drawn at each data point.</param>
00469 /// <param name="swarmPointSize">The size of the symbol drawn at each data point.</param>
00470 /// <param name="swarmPointMargin">The margin between data point symbols.  Set this to a negative
```

```
value to compress the plot, or to a positive value to expand it.</param>
00471 /// <param name="swarmPresentationAttributes">Presentation attributes for the swarm data
points.</param>
00472 /// <param name="boxPresentationAttributes">Presentation attributes for the box plots.</param>
00473 /// <param name="medianPresentationAttributes">Presentation attributes for the symbol drawn at the
medians.</param>
00474 /// <param name="medianSymbol">The symbol drawn at the medians.</param>
00475 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
to plot space.</param>
00476 /// <returns>A <see cref="Plot"/> containing the swarm plot.</returns>
00477             public static Plot SwarmPlot(IReadOnlyList<double> data, bool showBoxPlots = true, double
boxWidth = 0.04, double spacing = 10, double?  dataRangeMin = null, double?  dataRangeMax = null, bool
vertical = true, double width = 350, double height = 250,
00478                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00479                 double axisArrowSize = 3,
00480                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00481                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00482                 string xAxisTitle = null,
00483                 string yAxisTitle = null,
00484                 string title = null,
00485                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00486                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00487                 IDataPointElement swarmSymbol = null,
00488                 double swarmPointSize = 2,
00489                 double swarmPointMargin = 0.25,
00490                 IReadOnlyList<PlotElementPresentationAttributes> swarmPresentationAttributes = null,
00491                 IReadOnlyList<PlotElementPresentationAttributes> boxPresentationAttributes = null,
00492                 IReadOnlyList<PlotElementPresentationAttributes> medianPresentationAttributes = null,
00493                 IDataPointElement medianSymbol = null,
00494                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00495             {
00496                 return SwarmPlot(new (string, IReadOnlyList<double>)[] { (null, data) }, showBoxPlots,
boxWidth, spacing, dataRangeMin, dataRangeMax, vertical, width, height, axisPresentationAttributes,
axisArrowSize, axisLabelPresentationAttributes, axisTitlePresentationAttributes, xAxisTitle,
yAxisTitle, title, titlePresentationAttributes, gridPresentationAttributes, swarmSymbol,
swarmPointSize, swarmPointMargin, swarmPresentationAttributes, boxPresentationAttributes,
medianPresentationAttributes, medianSymbol, coordinateSystem);
00497             }
00498         }
00499     }
00500 }
```

8.54 Plot.ViolinPlot.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using MathNet.Numerics.Statistics;
00019 using System;
00020 using System.Collections.Generic;
00021 using System.Linq;
00022
00023 namespace VectSharp.Plots
00024 {
00025     partial class Plot
00026     {
00027         partial class Create
00028         {
00029 /// <summary>
00030 /// Create a new violin plot.
00031 /// </summary>
00032 /// <param name="data">The data to plot.</param>
00033 /// <param name="vertical">If this is <see langword="true"/> (the default), the violins are parallel
to the Y axis.  If this is <see langword="false"/>, the violins are parallel to the X axis.</param>
00034 /// <param name="smooth">If this is <see langword="true"/> (the default), the violin is smoothed.  If
this is <see langword="false"/>, it is not smoothed.</param>
00035 /// <param name="sides">Determines on which side(s) the violins are drawn.</param>
00036 /// <param name="showBoxPlots">If this is <see langword="true"/> (the default), a box plot is drawn on
top of each violin plot.</param>
```

```
00037 /// <param name="proportionalWidth">If this is <see langword="false"/> (the default), all the violins
      have the same width.  Otherwise, the width of each violin is proportional to the number of
      samples.</param>
00038 /// <param name="violinWidth">The width of the violins in data space coordinates.</param>
00039 /// <param name="boxWidth">The width of box plots as a proportion of the width of the violins.</param>
00040 /// <param name="spacing">The spacing between consecutive violins (ranging from 0 to 1).</param>
00041 /// <param name="dataRangeMin">If this is not <see langword="null"/>, this value is used to override
      the default minimum value for the plot.  Useful if you wish to create multiple plots with the same
      scale,
00042 /// even though the sampled values have different ranges.</param>
00043 /// <param name="dataRangeMax">If this is not <see langword="null"/>, this value is used to override
      the default maximum value for the plot.  Useful if you wish to create multiple plots with the same
      scale,
00044 /// even though the sampled values have different ranges.</param>
00045 /// <param name="width">The width of the plot.</param>
00046 /// <param name="height">The height of the plot.</param>
00047 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00048 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00049 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00050 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00051 /// <param name="xAxisTitle">Title for the X axis.</param>
00052 /// <param name="yAxisTitle">Title for the Y axis.</param>
00053 /// <param name="title">Title for the plot.</param>
00054 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00055 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00056 /// <param name="violinPresentationAttributes">Presentation attributes for the violins.</param>
00057 /// <param name="boxPresentationAttributes">Presentation attributes for the box plots.</param>
00058 /// <param name="medianPresentationAttributes">Presentation attributes for the symbol drawn at the
      medians.</param>
00059 /// <param name="medianSymbol">The symbol drawn at the medians.</param>
00060 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00061 /// <returns>A <see cref="Plot"/> containing the violin plot.</returns>
00062             public static Plot ViolinPlot<T>(IReadOnlyList<(T, IReadOnlyList<double>)> data, bool
      proportionalWidth = false, bool smooth = true, Violin.ViolinSide sides = Violin.ViolinSide.Both, bool
      showBoxPlots = true, double violinWidth = 10, double boxWidth = 0.05, double spacing = 0.1, double?
      dataRangeMin = null, double?  dataRangeMax = null, bool vertical = true, double width = 350, double
      height = 250,
00063                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00064                 double axisArrowSize = 3,
00065                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00066                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00067                 string xAxisTitle = null,
00068                 string yAxisTitle = null,
00069                 string title = null,
00070                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00071                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00072                 IReadOnlyList<PlotElementPresentationAttributes> violinPresentationAttributes = null,
00073                 IReadOnlyList<PlotElementPresentationAttributes> boxPresentationAttributes = null,
00074                 IReadOnlyList<PlotElementPresentationAttributes> medianPresentationAttributes = null,
00075                 IDataPointElement medianSymbol = null,
00076                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00077             {
00078                 if (violinPresentationAttributes == null)
00079                 {
00080                     violinPresentationAttributes = new PlotElementPresentationAttributes[] { new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(197, 235, 255)),
      Stroke = new SolidColourBrush(Colour.FromRgb(0, 114, 178)) },
00081                                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 233, 218)),
      Stroke = new SolidColourBrush(Colour.FromRgb(213, 94, 0)) },
00082                                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 222, 240)),
      Stroke = new SolidColourBrush(Colour.FromRgb(204, 121, 167)) },
00083                                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 242, 216)),
      Stroke = new SolidColourBrush(Colour.FromRgb(230, 159, 0)) },
00084                                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(214, 241, 255)),
      Stroke = new SolidColourBrush(Colour.FromRgb(86, 180, 233)) },
00085                                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(203, 255, 239)),
      Stroke = new SolidColourBrush(Colour.FromRgb(0, 158, 115)) },
00086                                                                                                 new
      PlotElementPresentationAttributes() { Fill = new SolidColourBrush(Colour.FromRgb(255, 249, 189)),
      Stroke = new SolidColourBrush(Colour.FromRgb(240, 228, 66)) } };
00087                 }
00088
00089                 if (boxPresentationAttributes == null)
00090                 {
00091                     PlotElementPresentationAttributes[] temp = new
      PlotElementPresentationAttributes[violinPresentationAttributes.Count];
00092
00093                     for (int i = 0; i < violinPresentationAttributes.Count; i++)
00094                     {
00095                         temp[i] = new PlotElementPresentationAttributes() { Stroke =
      violinPresentationAttributes[i].Stroke, Fill = violinPresentationAttributes[i].Stroke };
```

```
00096                     }
00097
00098                     boxPresentationAttributes = temp;
00099                 }
00100
00101                 if (medianPresentationAttributes == null)
00102                 {
00103                     PlotElementPresentationAttributes[] temp = new
      PlotElementPresentationAttributes[violinPresentationAttributes.Count];
00104
00105                     for (int i = 0; i < violinPresentationAttributes.Count; i++)
00106                     {
00107                         temp[i] = new PlotElementPresentationAttributes() { Stroke = null, Fill =
      violinPresentationAttributes[i].Fill };
00108                     }
00109
00110                     medianPresentationAttributes = temp;
00111                 }
00112
00113                 if (medianSymbol == null)
00114                 {
00115                     medianSymbol = new PathDataPointElement();
00116                 }
00117
00118                 if (axisPresentationAttributes == null)
00119                 {
00120                     axisPresentationAttributes = new PlotElementPresentationAttributes();
00121                 }
00122
00123                 if (gridPresentationAttributes == null)
00124                 {
00125                     gridPresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      new SolidColourBrush(Colour.FromRgb(220, 220, 220)) };
00126                 }
00127
00128                 if (axisLabelPresentationAttributes == null)
00129                 {
00130                     axisLabelPresentationAttributes = new PlotElementPresentationAttributes() { Stroke
      = null };
00131                 }
00132
00133                 if (axisTitlePresentationAttributes == null)
00134                 {
00135                     axisTitlePresentationAttributes = new PlotElementPresentationAttributes() { Font =
      new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 14), Stroke =
      null };
00136                 }
00137
00138                 if (titlePresentationAttributes == null)
00139                 {
00140                     titlePresentationAttributes = new PlotElementPresentationAttributes() { Stroke =
      null, Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), 18)
      };
00141                 }
00142
00143                 double[] medians = new double[data.Count];
00144                 double[] firstQuartiles = new double[data.Count];
00145                 double[] thirdQuartiles = new double[data.Count];
00146                 double[] upperWhiskers = new double[data.Count];
00147                 double[] lowerWhiskers = new double[data.Count];
00148                 double[][] centredData = new double[data.Count][];
00149
00150                 double minData = double.MaxValue;
00151                 double maxData = double.MinValue;
00152
00153                 for (int i = 0; i < data.Count; i++)
00154                 {
00155                     medians[i] = data[i].Item2.Median();
00156                     firstQuartiles[i] = data[i].Item2.LowerQuartile();
00157                     thirdQuartiles[i] = data[i].Item2.UpperQuartile();
00158
00159
00160                     upperWhiskers[i] = thirdQuartiles[i] + (thirdQuartiles[i] - firstQuartiles[i]) *
      1.5;
00161                     lowerWhiskers[i] = firstQuartiles[i] - (thirdQuartiles[i] - firstQuartiles[i]) *
      1.5;
00162
00163                     double uW = double.MinValue;
00164                     double lW = double.MaxValue;
00165
00166                     for (int j = 0; j < data[i].Item2.Count; j++)
00167                     {
00168                         if (data[i].Item2[j] <= upperWhiskers[i])
00169                         {
00170                             uW = Math.Max(uW, data[i].Item2[j]);
00171                         }
00172
```

```
00173                         if (data[i].Item2[j] >= lowerWhiskers[i])
00174                         {
00175                             lW = Math.Min(lW, data[i].Item2[j]);
00176                         }
00177 
00178                         minData = Math.Min(minData, data[i].Item2[j]);
00179                         maxData = Math.Max(maxData, data[i].Item2[j]);
00180                     }
00181 
00182                     upperWhiskers[i] = uW;
00183                     lowerWhiskers[i] = lW;
00184 
00185                     centredData[i] = new double[data[i].Item2.Count];
00186 
00187                     for (int j = 0; j < data[i].Item2.Count; j++)
00188                     {
00189                         centredData[i][j] = data[i].Item2[j] - medians[i];
00190                     }
00191                 }
00192 
00193                 double minX, maxX, minY, maxY;
00194 
00195                 if (vertical)
00196                 {
00197                     minX = 0;
00198                     maxX = violinWidth * (data.Count + spacing * (data.Count - 1));
00199 
00200                     minY = minData;
00201                     maxY = maxData;
00202 
00203                     if (dataRangeMin != null)
00204                     {
00205                         minY = dataRangeMin.Value;
00206                     }
00207 
00208                     if (dataRangeMax != null)
00209                     {
00210                         maxY = dataRangeMax.Value;
00211                     }
00212                 }
00213                 else
00214                 {
00215                     minY = 0;
00216                     maxY = violinWidth * (data.Count + spacing * (data.Count - 1));
00217 
00218                     minX = minData;
00219                     maxX = maxData;
00220 
00221                     if (dataRangeMin != null)
00222                     {
00223                         minX = dataRangeMin.Value;
00224                     }
00225 
00226                     if (dataRangeMax != null)
00227                     {
00228                         maxX = dataRangeMax.Value;
00229                     }
00230                 }
00231 
00232 
00233                 if (coordinateSystem == null)
00234                 {
00235                     coordinateSystem = new LinearCoordinateSystem2D(minX, maxX, minY, maxY, width,
      height);
00236                 }
00237 
00238                 Point topLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, maxY });
00239                 Point topRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, maxY });
00240                 Point bottomRight = coordinateSystem.ToPlotCoordinates(new double[] { maxX, minY });
00241                 Point bottomLeft = coordinateSystem.ToPlotCoordinates(new double[] { minX, minY });
00242 
00243                 double[] marginTopLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00244                 double[] marginTopRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00245                 double[] marginBottomRight = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00246                 double[] marginBottomLeft = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00247 
00248                 double[] p1 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
```

```
00249                 double[] p2 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y)) - 10));
00250                 double[] p3 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00251                 double[] p4 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)),
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y)) + 10));
00252
00253
00254                 double[] p5 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00255                 double[] p6 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X, bottomRight.X)) - 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00256                 double[] p7 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y, bottomRight.Y))));
00257                 double[] p8 = coordinateSystem.ToDataCoordinates(new
      Point(Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X, bottomRight.X)) + 10,
      Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y, bottomRight.Y))));
00258
00259                 ContinuousAxis xAxis = new ContinuousAxis(marginBottomLeft[0] < marginBottomRight[0] ?
      marginBottomLeft :  marginBottomRight, marginBottomLeft[0] < marginBottomRight[0] ?  marginBottomRight
      :  vertical ?  marginBottomLeft :
      coordinateSystem.ToDataCoordinates(coordinateSystem.ToPlotCoordinates(marginBottomLeft) + new
      Point(-axisArrowSize - 7, 0)), coordinateSystem) { PresentationAttributes =
      axisPresentationAttributes, ArrowSize = vertical ?  0 :  axisArrowSize };
00260                 ContinuousAxis yAxis = new ContinuousAxis(marginBottomLeft[1] < marginTopLeft[1] ?
      marginBottomLeft :  marginTopLeft, marginBottomLeft[1] < marginTopLeft[1] ?  marginTopLeft :
      !vertical ?  marginBottomLeft :
      coordinateSystem.ToDataCoordinates(coordinateSystem.ToPlotCoordinates(marginBottomLeft) + new Point(0,
      axisArrowSize + 7)), coordinateSystem) { PresentationAttributes = axisPresentationAttributes,
      ArrowSize = vertical ?  axisArrowSize :  0 };
00261
00262                 IPlotElement xTicks;
00263                 IPlotElement yTicks;
00264
00265                 IPlotElement xLabels;
00266                 IPlotElement yLabels;
00267
00268                 if (vertical)
00269                 {
00270                     xTicks = new ScatterPoints<IReadOnlyList<double>>((from el in Enumerable.Range(0,
      data.Count) select new double[] { (el * (spacing + 1) + 0.5) * violinWidth, p3[1] }),
      coordinateSystem) { PresentationAttributes = axisPresentationAttributes, Size = 1, DataPointElement =
      new PathDataPointElement(new GraphicsPath().MoveTo(0, -3).LineTo(0, 3)) };
00271                     yTicks = new ContinuousAxisTicks(p6, p5, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00272
00273                     xLabels = new DataLabels<IReadOnlyList<double>>((from el in Enumerable.Range(0,
      data.Count) select new double[] { (el * (spacing + 1) + 0.5) * violinWidth, p3[1] }),
      coordinateSystem) { Alignment = TextAnchors.Center, Baseline = TextBaselines.Top, Label = (i, _) =>
      data[i].Item1, PresentationAttributes = axisLabelPresentationAttributes, Margin = (a, b) => new
      Point(0, 10) };
00274                     yLabels = new ContinuousAxisLabels(p6, p5, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Position = _ => -10, Alignment =
      TextAnchors.Right, Rotation = 0, IntervalCount = 5 };
00275                 }
00276                 else
00277                 {
00278                     xTicks = new ContinuousAxisTicks(p3, p4, coordinateSystem) {
      PresentationAttributes = axisPresentationAttributes };
00279
00280                     yTicks = new ScatterPoints<IReadOnlyList<double>>((from el in Enumerable.Range(0,
      data.Count) select new double[] { p6[0], (el * (spacing + 1) + 0.5) * violinWidth }),
      coordinateSystem) { PresentationAttributes = axisPresentationAttributes, Size = 1, DataPointElement =
      new PathDataPointElement(new GraphicsPath().MoveTo(-3, 0).LineTo(3, 0)) };
00281
00282                     xLabels = new ContinuousAxisLabels(p3, p4, coordinateSystem) {
      PresentationAttributes = axisLabelPresentationAttributes, Alignment = TextAnchors.Center, Baseline =
      TextBaselines.Top, Rotation = 0, IntervalCount = 5 };
00283                     yLabels = new DataLabels<IReadOnlyList<double>>((from el in Enumerable.Range(0,
      data.Count) select new double[] { p6[0], (el * (spacing + 1) + 0.5) * violinWidth }),
      coordinateSystem) { Alignment = TextAnchors.Right, Baseline = TextBaselines.Middle, Label = (i, _) =>
      data[i].Item1, PresentationAttributes = axisLabelPresentationAttributes, Margin = (a, b) => new
      Point(-10, 0) };
00284                 }
00285
00286                 Graphics xLabelsSize = new Graphics();
00287                 xLabels.Plot(xLabelsSize);
00288                 double xLabelsHeight = xLabelsSize.GetBounds().Size.Height;
00289
00290                 Graphics yLabelsSize = new Graphics();
00291                 yLabels.Plot(yLabelsSize);
```

```
00292                 double yLabelsWidth = yLabelsSize.GetBounds().Size.Width;
00293
00294                 ContinuousAxisTitle xTitle = new ContinuousAxisTitle(xAxisTitle, marginBottomLeft,
      marginBottomRight, coordinateSystem, axisTitlePresentationAttributes) { Position = xLabelsHeight + 20,
      Alignment = TextAnchors.Center };
00295                 ContinuousAxisTitle yTitle = new ContinuousAxisTitle(yAxisTitle, marginBottomLeft,
      marginTopLeft, coordinateSystem, axisTitlePresentationAttributes) { Position = -20 - yLabelsWidth,
      Baseline = TextBaselines.Bottom, Alignment = TextAnchors.Center };
00296
00297                 Plot plot = new Plot();
00298
00299                 if (vertical)
00300                 {
00301                     Grid yGrid = new Grid(p5, p6, p7, p8, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00302                     plot.AddPlotElement(yGrid);
00303                 }
00304                 else
00305                 {
00306                     Grid xGrid = new Grid(p1, p2, p3, p4, coordinateSystem) { IntervalCount = 5,
      PresentationAttributes = gridPresentationAttributes };
00307                     plot.AddPlotElement(xGrid);
00308                 }
00309
00310                 plot.AddPlotElements(xAxis, yAxis, xTicks, yTicks, xLabels, yLabels, xTitle, yTitle);
00311
00312                 double maxCount = (from el in data select el.Item2.Count).Max();
00313
00314                 for (int i = 0; i < data.Count; i++)
00315                 {
00316                     if (vertical)
00317                     {
00318                         Violin violin = new Violin(new double[] { (i * (spacing + 1) + 0.5) *
      violinWidth, medians[i] }, new double[] { 0, 1 }, centredData[i], coordinateSystem)
00319                         {
00320                             PresentationAttributes = violinPresentationAttributes[i %
      boxPresentationAttributes.Count],
00321                             Width = proportionalWidth ?  (violinWidth * 0.5 * data[i].Item2.Count /
      maxCount) :  (violinWidth * 0.5),
00322                             Smooth = smooth,
00323                             Sides = sides
00324                         };
00325                         plot.AddPlotElement(violin);
00326
00327                         if (showBoxPlots)
00328                         {
00329                             BoxPlot box = new BoxPlot(new double[] { (i * (spacing + 1) + 0.5) *
      violinWidth, medians[i] }, new double[] { 0, 1 }, lowerWhiskers[i] - medians[i], firstQuartiles[i] -
      medians[i], thirdQuartiles[i] - medians[i], upperWhiskers[i] - medians[i], coordinateSystem)
00330                             {
00331                                 BoxPresentationAttributes = boxPresentationAttributes[i %
      boxPresentationAttributes.Count],
00332                                 WhiskersPresentationAttributes = boxPresentationAttributes[i %
      boxPresentationAttributes.Count],
00333                                 Width = (proportionalWidth ?  (violinWidth * 0.5 * data[i].Item2.Count
      / maxCount) :  (violinWidth * 0.5)) * boxWidth,
00334                                 NotchSize = 0,
00335                                 WhiskerWidth = 0,
00336                                 CentreSymbolPresentationAttributes = medianPresentationAttributes[i %
      medianPresentationAttributes.Count],
00337                                 CentreSymbol = medianSymbol
00338                             };
00339
00340                             plot.AddPlotElement(box);
00341                         }
00342                     }
00343                     else
00344                     {
00345                         Violin violin = new Violin(new double[] { medians[i], (i * (spacing + 1) +
      0.5) * violinWidth }, new double[] { 1, 0 }, centredData[i], coordinateSystem)
00346                         {
00347                             PresentationAttributes = violinPresentationAttributes[i %
      boxPresentationAttributes.Count],
00348                             Width = proportionalWidth ?  (violinWidth * 0.5 * data[i].Item2.Count /
      maxCount) :  (violinWidth * 0.5),
00349                             Smooth = smooth,
00350                             Sides = sides
00351                         };
00352                         plot.AddPlotElement(violin);
00353
00354                         if (showBoxPlots)
00355                         {
00356                             BoxPlot box = new BoxPlot(new double[] { medians[i], (i * (spacing + 1) +
      0.5) * violinWidth }, new double[] { 1, 0 }, lowerWhiskers[i] - medians[i], firstQuartiles[i] -
      medians[i], thirdQuartiles[i] - medians[i], upperWhiskers[i] - medians[i], coordinateSystem)
00357                             {
00358                                 BoxPresentationAttributes = boxPresentationAttributes[i %
```

```
boxPresentationAttributes.Count],
00359                                 WhiskersPresentationAttributes = boxPresentationAttributes[i %
boxPresentationAttributes.Count],
00360                                 Width = (proportionalWidth ?  (violinWidth * 0.5 * data[i].Item2.Count
/ maxCount) :  (violinWidth * 0.5)) * boxWidth,
00361                                 NotchSize = 0,
00362                                 WhiskerWidth = 0,
00363                                 CentreSymbolPresentationAttributes = medianPresentationAttributes[i %
medianPresentationAttributes.Count],
00364                                 CentreSymbol = medianSymbol
00365                             };
00366
00367                             plot.AddPlotElement(box);
00368                         }
00369                     }
00370                 }
00371
00372                 TextLabel<IReadOnlyList<double» titleLabel = new
TextLabel<IReadOnlyList<double»(title,
coordinateSystem.ToDataCoordinates((coordinateSystem.ToPlotCoordinates(marginTopLeft) +
coordinateSystem.ToPlotCoordinates(marginTopRight)) * 0.5 + new Point(0, -10)), coordinateSystem) {
Baseline = TextBaselines.Bottom, PresentationAttributes = titlePresentationAttributes };
00373
00374                 plot.AddPlotElement(titleLabel);
00375
00376                 return plot;
00377             }
00378
00379 /// <summary>
00380 /// Create a new violin plot.
00381 /// </summary>
00382 /// <param name="data">The data to plot.</param>
00383 /// <param name="vertical">If this is <see langword="true"/> (the default), the violins are parallel
to the Y axis.  If this is <see langword="false"/>, the violins are parallel to the X axis.</param>
00384 /// <param name="smooth">If this is <see langword="true"/> (the default), the violin is smoothed.  If
this is <see langword="false"/>, it is not smoothed.</param>
00385 /// <param name="sides">Determines on which side(s) the violins are drawn.</param>
00386 /// <param name="showBoxPlots">If this is <see langword="true"/> (the default), a box plot is drawn on
top of each violin plot.</param>
00387 /// <param name="proportionalWidth">If this is <see langword="false"/> (the default), all the violins
have the same width.  Otherwise, the width of each violin is proportional to the number of
samples.</param>
00388 /// <param name="violinWidth">The width of the violins in data space coordinates.</param>
00389 /// <param name="boxWidth">The width of box plots as a proportion of the width of the violins.</param>
00390 /// <param name="spacing">The spacing between consecutive violins (ranging from 0 to 1).</param>
00391 /// <param name="dataRangeMin">If this is not <see langword="null"/>, this value is used to override
the default minimum value for the plot.  Useful if you wish to create multiple plots with the same
scale,
00392 /// even though the sampled values have different ranges.</param>
00393 /// <param name="dataRangeMax">If this is not <see langword="null"/>, this value is used to override
the default maximum value for the plot.  Useful if you wish to create multiple plots with the same
scale,
00394 /// even though the sampled values have different ranges.</param>
00395 /// <param name="width">The width of the plot.</param>
00396 /// <param name="height">The height of the plot.</param>
00397 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00398 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00399 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00400 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00401 /// <param name="xAxisTitle">Title for the X axis.</param>
00402 /// <param name="yAxisTitle">Title for the Y axis.</param>
00403 /// <param name="title">Title for the plot.</param>
00404 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00405 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00406 /// <param name="violinPresentationAttributes">Presentation attributes for the violins.</param>
00407 /// <param name="boxPresentationAttributes">Presentation attributes for the box plots.</param>
00408 /// <param name="medianPresentationAttributes">Presentation attributes for the symbol drawn at the
medians.</param>
00409 /// <param name="medianSymbol">The symbol drawn at the medians.</param>
00410 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
to plot space.</param>
00411 /// <returns>A <see cref="Plot"/> containing the violin plot.</returns>
00412             public static Plot ViolinPlot(IReadOnlyList<IReadOnlyList<double» data, bool
proportionalWidth = false, bool smooth = true, Violin.ViolinSide sides = Violin.ViolinSide.Both, bool
showBoxPlots = true, double violinWidth = 10, double boxWidth = 0.05, double spacing = 0.1, double?
dataRangeMin = null, double?  dataRangeMax = null, bool vertical = true, double width = 350, double
height = 250,
00413                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00414                 double axisArrowSize = 3,
00415                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00416                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00417                 string xAxisTitle = null,
00418                 string yAxisTitle = null,
00419                 string title = null,
00420                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00421                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00422                 IReadOnlyList<PlotElementPresentationAttributes> violinPresentationAttributes = null,
```

```
00423                 IReadOnlyList<PlotElementPresentationAttributes> boxPresentationAttributes = null,
00424                 IReadOnlyList<PlotElementPresentationAttributes> medianPresentationAttributes = null,
00425                 IDataPointElement medianSymbol = null,
00426                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00427             {
00428                 return ViolinPlot((from el in data select ((string)null, el)).ToArray(),
      proportionalWidth, smooth, sides, showBoxPlots, violinWidth, boxWidth, spacing, dataRangeMin,
      dataRangeMax, vertical, width, height, axisPresentationAttributes, axisArrowSize,
      axisLabelPresentationAttributes, axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title,
      titlePresentationAttributes, gridPresentationAttributes, violinPresentationAttributes,
      boxPresentationAttributes, medianPresentationAttributes, medianSymbol, coordinateSystem);
00429             }
00430
00431 /// <summary>
00432 /// Create a new violin plot.
00433 /// </summary>
00434 /// <param name="data">The data to plot.</param>
00435 /// <param name="vertical">If this is <see langword="true"/> (the default), the violin is parallel to
      the Y axis.  If this is <see langword="false"/>, the violin is parallel to the X axis.</param>
00436 /// <param name="smooth">If this is <see langword="true"/> (the default), the violin is smoothed.  If
      this is <see langword="false"/>, it is not smoothed.</param>
00437 /// <param name="sides">Determines on which side(s) the violin is drawn.</param>
00438 /// <param name="showBoxPlots">If this is <see langword="true"/> (the default), a box plot is drawn on
      top of the violin plot.</param>
00439 /// <param name="violinWidth">The width of the violin in data space coordinates.</param>
00440 /// <param name="boxWidth">The width of box plot as a proportion of the width of the violin.</param>
00441 /// <param name="dataRangeMin">If this is not <see langword="null"/>, this value is used to override
      the default minimum value for the plot.  Useful if you wish to create multiple plots with the same
      scale,
00442 /// even though the sampled values have different ranges.</param>
00443 /// <param name="dataRangeMax">If this is not <see langword="null"/>, this value is used to override
      the default maximum value for the plot.  Useful if you wish to create multiple plots with the same
      scale,
00444 /// even though the sampled values have different ranges.</param>
00445 /// <param name="width">The width of the plot.</param>
00446 /// <param name="height">The height of the plot.</param>
00447 /// <param name="axisPresentationAttributes">Presentation attributes for the axes.</param>
00448 /// <param name="axisArrowSize">Size of the arrow at the end of each axis.</param>
00449 /// <param name="axisLabelPresentationAttributes">Presentation attributes for the axis labels.</param>
00450 /// <param name="axisTitlePresentationAttributes">Presentation attributes for the axis titles.</param>
00451 /// <param name="xAxisTitle">Title for the X axis.</param>
00452 /// <param name="yAxisTitle">Title for the Y axis.</param>
00453 /// <param name="title">Title for the plot.</param>
00454 /// <param name="titlePresentationAttributes">Presentation attributes for the plot title.</param>
00455 /// <param name="gridPresentationAttributes">Presentation attributes for the grid.</param>
00456 /// <param name="violinPresentationAttributes">Presentation attributes for the violin.</param>
00457 /// <param name="boxPresentationAttributes">Presentation attributes for the box plots.</param>
00458 /// <param name="medianPresentationAttributes">Presentation attributes for the symbol drawn at the
      medians.</param>
00459 /// <param name="medianSymbol">The symbol drawn at the medians.</param>
00460 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00461 /// <returns>A <see cref="Plot"/> containing the violin plot.</returns>
00462             public static Plot ViolinPlot(IReadOnlyList<double> data, bool smooth = true,
      Violin.ViolinSide sides = Violin.ViolinSide.Both, bool showBoxPlots = true, double violinWidth = 10,
      double boxWidth = 0.05, double?  dataRangeMin = null, double?  dataRangeMax = null, bool vertical =
      true, double width = 350, double height = 250,
00463                 PlotElementPresentationAttributes axisPresentationAttributes = null,
00464                 double axisArrowSize = 3,
00465                 PlotElementPresentationAttributes axisLabelPresentationAttributes = null,
00466                 PlotElementPresentationAttributes axisTitlePresentationAttributes = null,
00467                 string xAxisTitle = null,
00468                 string yAxisTitle = null,
00469                 string title = null,
00470                 PlotElementPresentationAttributes titlePresentationAttributes = null,
00471                 PlotElementPresentationAttributes gridPresentationAttributes = null,
00472                 PlotElementPresentationAttributes violinPresentationAttributes = null,
00473                 PlotElementPresentationAttributes boxPresentationAttributes = null,
00474                 PlotElementPresentationAttributes medianPresentationAttributes = null,
00475                 IDataPointElement medianSymbol = null,
00476                 IContinuousInvertibleCoordinateSystem coordinateSystem = null)
00477             {
00478                 return ViolinPlot(new (string, IReadOnlyList<double>)[] { (null, data) }, false,
      smooth, sides, showBoxPlots, violinWidth, boxWidth, 0, dataRangeMin, dataRangeMax, vertical, width,
      height, axisPresentationAttributes, axisArrowSize, axisLabelPresentationAttributes,
      axisTitlePresentationAttributes, xAxisTitle, yAxisTitle, title, titlePresentationAttributes,
      gridPresentationAttributes, violinPresentationAttributes == null ?  null :  new
      PlotElementPresentationAttributes[] { violinPresentationAttributes }, boxPresentationAttributes ==
      null ?  null :  new PlotElementPresentationAttributes[] { boxPresentationAttributes },
      medianPresentationAttributes == null ?  null :  new PlotElementPresentationAttributes[] {
      medianPresentationAttributes }, medianSymbol, coordinateSystem);
00479             }
00480         }
00481     }
00482 }
```

8.55 Swarm.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2024 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021
00022 namespace VectSharp.Plots
00023 {
00024 /// <summary>
00025 /// A plot element that draws a swarm plot.
00026 /// </summary>
00027     public class Swarm :  IPlotElement
00028     {
00029 /// <summary>
00030 /// The position of the origin of the swarm (e.g., the 0 in data space coordinates).
00031 /// </summary>
00032         public IReadOnlyList<double> Position { get; set; }
00033
00034 /// <summary>
00035 /// The direction along which the swarm is drawn, in data space coordinates.
00036 /// </summary>
00037         public IReadOnlyList<double> Direction { get; set; }
00038
00039 /// <summary>
00040 /// The values whose distribution is displayed by the swarm plot.
00041 /// </summary>
00042         public IReadOnlyList<double> Data { get; }
00043
00044 /// <summary>
00045 /// The size of the points in the swarm plot, in plot coordinates.
00046 /// </summary>
00047         public double PointSize { get; set; } = 2;
00048
00049 /// <summary>
00050 /// The minimum margin between points in the swarm plot, in plot coordinates.
00051 /// </summary>
00052         public double PointMargin { get; set; } = 0.25;
00053
00054 /// <summary>
00055 /// The <see cref="IDataPointElement"/> used to draw the swarm points.
00056 /// </summary>
00057         public IDataPointElement SwarmPointElement { get; set; } = new PathDataPointElement();
00058
00059 /// <summary>
00060 /// Presentation attributes for the swarm plot.
00061 /// </summary>
00062         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes() { Fill = Colours.White };
00063
00064 /// <summary>
00065 /// The coordinate system used to transform the points from data space to plot space.
00066 /// </summary>
00067         public IContinuousInvertibleCoordinateSystem CoordinateSystem { get; set; }
00068         ICoordinateSystem IPlotElement.CoordinateSystem => this.CoordinateSystem;
00069
00070 /// <summary>
00071 /// A tag to identify the swarm in the plot.
00072 /// </summary>
00073         public string Tag { get; set; }
00074
00075 /// <summary>
00076 /// Create a new <see cref="Swarm"/> instance.
00077 /// </summary>
00078 /// <param name="position">The position of the origin of the swarm (e.g., the 0 in data space
      coordinates).</param>
00079 /// <param name="direction">The direction along which the swarm is drawn, in data space
      coordinates.</param>
00080 /// <param name="data">The values whose distribution is displayed by the swarm plot.</param>
00081 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
```

```
00082         public Swarm(IReadOnlyList<double> position, IReadOnlyList<double> direction,
      IEnumerable<double> data, IContinuousInvertibleCoordinateSystem coordinateSystem)
00083         {
00084             this.Position = position;
00085             this.Direction = direction;
00086             this.Data = data.OrderBy(x => x).ToArray();
00087             this.CoordinateSystem = coordinateSystem;
00088         }
00089 
00090         private static (Point, double) GetMinDistance(Point pt, List<Point> plottedPoints)
00091         {
00092             double minDist = double.MaxValue;
00093             Point minPoint = new Point();
00094 
00095             for (int i = 0; i < plottedPoints.Count; i++)
00096             {
00097                 double dist = (plottedPoints[i] - pt).Modulus();
00098                 if (dist < minDist)
00099                 {
00100                     minDist = dist;
00101                     minPoint = plottedPoints[i];
00102                 }
00103             }
00104 
00105             return (minPoint, minDist);
00106         }
00107 
00108         private IReadOnlyList<double> ProposePoint(IReadOnlyList<double> pt, List<Point>
      plottedPoints, IReadOnlyList<double> perpendicularDirection)
00109         {
00110             Point projPt = CoordinateSystem.ToPlotCoordinates(pt);
00111 
00112             (Point closestPoint, double minDist) = GetMinDistance(projPt, plottedPoints);
00113 
00114             while (minDist < PointSize * 2 + PointMargin * PointSize)
00115             {
00116                 Point localPerpDirectionPt =
      (CoordinateSystem.ToPlotCoordinates(CoordinateSystem.GetAround(CoordinateSystem.ToDataCoordinates(projPt),
      perpendicularDirection)) - projPt).Normalize();
00117                 double parallDist = (projPt - closestPoint).Select((x, i) => x *
      localPerpDirectionPt[i]).Sum();
00118                 projPt = projPt + localPerpDirectionPt * (-parallDist + Math.Sqrt(parallDist *
      parallDist - (projPt - closestPoint).Select(x => x * x).Sum() + (PointSize * 2 + PointMargin *
      PointSize) * (PointSize * 2 + PointMargin * PointSize))) * 1.0001;
00119                 (closestPoint, minDist) = GetMinDistance(projPt, plottedPoints);
00120             }
00121 
00122             return CoordinateSystem.ToDataCoordinates(projPt);
00123         }
00124 
00125         private static double[] GetPerpendicularVector(IReadOnlyList<double> v)
00126         {
00127             double[] tbr = new double[v.Count];
00128 
00129             int m = -1;
00130 
00131             for (int i = 0; i < v.Count; i++)
00132             {
00133                 if (v[i] != 0)
00134                 {
00135                     m = i; break;
00136                 }
00137             }
00138 
00139             if (m < 0)
00140             {
00141                 return tbr;
00142             }
00143 
00144             int n = m == 0 ?  1 :  0;
00145 
00146             double mod = Math.Sqrt(v[m] * v[m] + v[n] * v[n]);
00147 
00148             tbr[m] = -v[n] / mod;
00149             tbr[n] = v[m] / mod;
00150 
00151             return tbr;
00152         }
00153 
00154 /// <inheritdoc/>
00155         public void Plot(Graphics target)
00156         {
00157             List<Point> plottedPoints = new List<Point>();
00158 
00159             double[] perpDirection1 = GetPerpendicularVector(this.Direction);
00160             double[] perpDirection2 = perpDirection1.Select(x => -x).ToArray();
00161 
```

```
00162             for (int i = 0; i < Data.Count; i++)
00163             {
00164                 double[] point = new double[this.Position.Count];
00165 
00166                 for (int j = 0; j < this.Position.Count; j++)
00167                 {
00168                     point[j] = this.Position[j] + Data[i] * this.Direction[j];
00169                 }
00170 
00171                 IReadOnlyList<double> point1 = ProposePoint(point, plottedPoints, perpDirection1);
00172                 IReadOnlyList<double> point2 = ProposePoint(point, plottedPoints, perpDirection2);
00173 
00174                 Point proj = CoordinateSystem.ToPlotCoordinates(point);
00175                 Point pt1 = CoordinateSystem.ToPlotCoordinates(point1);
00176                 Point pt2 = CoordinateSystem.ToPlotCoordinates(point2);
00177 
00178                 Point pt = (pt1 - proj).Modulus() < (pt2 - proj).Modulus() ?  pt1 :  pt2;
00179 
00180                 plottedPoints.Add(pt);
00181 
00182                 string tag = (Tag == null || !target.UseUniqueTags) ?  this.Tag :  (this.Tag + "@" +
      i.ToString());
00183 
00184                 target.Save();
00185                 target.Translate(pt);
00186                 target.Scale(PointSize, PointSize);
00187                 SwarmPointElement.Plot(target, this.PresentationAttributes, tag);
00188                 target.Restore();
00189             }
00190         }
00191     }
00192 }
```

8.56 Trendlines.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using MathNet.Numerics;
00019 using MathNet.Numerics.LinearAlgebra;
00020 using MathNet.Numerics.LinearAlgebra.Double;
00021 using MathNet.Numerics.LinearRegression;
00022 using System;
00023 using System.Collections.Generic;
00024 using System.Linq;
00025 
00026 namespace VectSharp.Plots
00027 {
00028 /// <summary>
00029 /// A plot element that draws a linear trendline with equation <c>y = a * x + b</c>.
00030 /// </summary>
00031     public class LinearTrendLine :  IPlotElement
00032     {
00033 /// <summary>
00034 /// The slope of the trendline (a).
00035 /// </summary>
00036         public double Slope { get; set; }
00037 
00038 /// <summary>
00039 /// The intercept of the trendline (b).
00040 /// </summary>
00041         public double Intercept { get; set; }
00042 
00043 /// <summary>
00044 /// The minimum X value for which the trendline is plotted.
00045 /// </summary>
00046         public double MinX { get; set; }
00047 
00048 /// <summary>
```

```
00049 /// The minimum Y value for which the trendline is plotted.
00050 /// </summary>
00051         public double MinY { get; set; }
00052
00053 /// <summary>
00054 /// The maximum X value for which the trendline is plotted.
00055 /// </summary>
00056         public double MaxX { get; set; }
00057
00058 /// <summary>
00059 /// The maximum Y value for which the trendline is plotted.
00060 /// </summary>
00061         public double MaxY { get; set; }
00062
00063 /// <summary>
00064 /// Presentation attributes for the trendline.
00065 /// </summary>
00066         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180,
      180, 180) };
00067
00068 /// <summary>
00069 /// A tag to identify the trendline in the plot.
00070 /// </summary>
00071         public string Tag { get; set; }
00072
00073 /// <summary>
00074 /// The coordinate system used to transform the points from data space to plot space.
00075 /// </summary>
00076         public IContinuousCoordinateSystem CoordinateSystem { get; set; }
00077         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00078
00079 /// <summary>
00080 /// Create a new <see cref="LinearTrendLine"/> instance, specifying the equation parameters.
00081 /// </summary>
00082 /// <param name="slope">The slope of the trendline (a).</param>
00083 /// <param name="intercept">The intercept of the trendline (b).</param>
00084 /// <param name="minX">The minimum X value for which the trendline is plotted.</param>
00085 /// <param name="minY">The minimum Y value for which the trendline is plotted.</param>
00086 /// <param name="maxX">The maximum X value for which the trendline is plotted.</param>
00087 /// <param name="maxY">The maximum Y value for which the trendline is plotted.</param>
00088 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00089         public LinearTrendLine(double slope, double intercept, double minX, double minY, double maxX,
      double maxY, IContinuousCoordinateSystem coordinateSystem)
00090         {
00091             Slope = slope;
00092             Intercept = intercept;
00093             MinX = minX;
00094             MinY = minY;
00095             MaxX = maxX;
00096             MaxY = maxY;
00097             CoordinateSystem = coordinateSystem;
00098         }
00099
00100 /// <summary>
00101 /// Create a new <see cref="LinearTrendLine"/> instance, determining the equation parameters by
      running a regression.
00102 /// </summary>
00103 /// <param name="data">The data that will be used to determine the equation parameters.</param>
00104 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00105 /// <param name="fixedIntercept">If this is <see langword="null"/>, the intercept (b) is determined
      during the regression; otherwise, it is fixed to the specified value.</param>
00106         public LinearTrendLine(IReadOnlyList<IReadOnlyList<double» data, IContinuousCoordinateSystem
      coordinateSystem, double?  fixedIntercept = null)
00107         {
00108             MinX = double.MaxValue;
00109             MinY = double.MaxValue;
00110             MaxX = double.MinValue;
00111             MaxY = double.MinValue;
00112
00113             double[] x = new double[data.Count];
00114             double[] y = new double[data.Count];
00115
00116             for (int i = 0; i < data.Count; i++)
00117             {
00118                 MinX = Math.Min(MinX, data[i][0]);
00119                 MinY = Math.Min(MinY, data[i][1]);
00120                 MaxX = Math.Max(MaxX, data[i][0]);
00121                 MaxY = Math.Max(MaxY, data[i][1]);
00122
00123                 x[i] = data[i][0];
00124
00125                 if (fixedIntercept == null)
00126                 {
00127                     y[i] = data[i][1];
```

```
00128                 }
00129                 else
00130                 {
00131                     y[i] = data[i][1] - fixedIntercept.Value;
00132                 }
00133             }
00134 
00135             if (fixedIntercept == null)
00136             {
00137                 (Intercept, Slope) = MathNet.Numerics.LinearRegression.SimpleRegression.Fit(x, y);
00138             }
00139             else
00140             {
00141                 Slope = MathNet.Numerics.LinearRegression.SimpleRegression.FitThroughOrigin(x, y);
00142                 Intercept = fixedIntercept.Value;
00143             }
00144 
00145             this.CoordinateSystem = coordinateSystem;
00146         }
00147 
00148 /// <summary>
00149 /// Create a new <see cref="LinearTrendLine"/> instance, determining the equation parameters by
      running a regression.
00150 /// </summary>
00151 /// <param name="data">The data that will be used to determine the equation parameters.</param>
00152 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00153 /// <param name="fixedIntercept">If this is <see langword="null"/>, the intercept (b) is determined
      during the regression; otherwise, it is fixed to the specified value.</param>
00154         public LinearTrendLine(IReadOnlyList<(double, double)> data, IContinuousCoordinateSystem
      coordinateSystem, double?  fixedIntercept = null) :  this((from el in data select new double[] {
      el.Item1, el.Item2 }).ToArray(), coordinateSystem, fixedIntercept) { }
00155 
00156 /// <inheritdoc/>
00157         public void Plot(Graphics target)
00158         {
00159             double y0 = Slope * MinX + Intercept;
00160             double x0 = (MinY - Intercept) / Slope;
00161 
00162             double y1 = Slope * MaxX + Intercept;
00163             double x1 = (MaxY - Intercept) / Slope;
00164 
00165             HashSet<(double, double)> points = new HashSet<(double, double)>();
00166 
00167             if (y0 >= MinY && y0 <= MaxY)
00168             {
00169                 points.Add((MinX, y0));
00170             }
00171 
00172             if (x0 >= MinX && x0 <= MaxX)
00173             {
00174                 points.Add((x0, MinY));
00175             }
00176 
00177             if (y1 >= MinY && y1 <= MaxY)
00178             {
00179                 points.Add((MaxX, y1));
00180             }
00181 
00182             if (x1 >= MinX && x1 <= MaxX)
00183             {
00184                 points.Add((x1, MaxY));
00185             }
00186 
00187             if (points.Count == 2)
00188             {
00189                 double[][] uniquePoints = new double[points.Count][];
00190 
00191                 int index = 0;
00192 
00193                 foreach ((double x, double y) in points)
00194                 {
00195                     uniquePoints[index] = new double[] { x, y };
00196                     index++;
00197                 }
00198 
00199                 GraphicsPath path;
00200 
00201                 if (CoordinateSystem.IsLinear || CoordinateSystem.IsDirectionStraight(new double[] {
      uniquePoints[1][0] - uniquePoints[0][0], uniquePoints[1][1] - uniquePoints[0][1] }))
00202                 {
00203                     Point p1 = CoordinateSystem.ToPlotCoordinates(uniquePoints[0]);
00204                     Point p2 = CoordinateSystem.ToPlotCoordinates(uniquePoints[1]);
00205 
00206                     path = new GraphicsPath().MoveTo(p1).LineTo(p2);
00207                 }
00208                 else
```

```
00209                 {
00210                     double[] direction = new double[] { uniquePoints[1][0] - uniquePoints[0][0],
      uniquePoints[1][1] - uniquePoints[0][1] };
00211                     double mod = Math.Sqrt(direction[0] * direction[0] + direction[1] * direction[1]);
00212                     double[] increment = new double[] { direction[0] / mod *
      Math.Min(CoordinateSystem.Resolution[0], CoordinateSystem.Resolution[1]), direction[1] / mod *
      Math.Min(CoordinateSystem.Resolution[0], CoordinateSystem.Resolution[1]) };
00213
00214                     int count = (int)Math.Ceiling(Math.Max(direction[0] / increment[0], direction[1] /
      increment[1]));
00215
00216                     path = new GraphicsPath();
00217
00218                     for (int i = 0; i <= count; i++)
00219                     {
00220                         double[] pt = new double[] { uniquePoints[0][0] + increment[0] * i,
      uniquePoints[0][1] + increment[1] * i };
00221
00222                         if (i == 0)
00223                         {
00224                             path.MoveTo(CoordinateSystem.ToPlotCoordinates(pt));
00225                         }
00226                         else
00227                         {
00228                             path.LineTo(CoordinateSystem.ToPlotCoordinates(pt));
00229                         }
00230                     }
00231                 }
00232
00233                 Point topLeft = CoordinateSystem.ToPlotCoordinates(new double[] { MinX, MinY });
00234                 Point topRight = CoordinateSystem.ToPlotCoordinates(new double[] { MaxX, MinY });
00235                 Point bottomRight = CoordinateSystem.ToPlotCoordinates(new double[] { MaxX, MaxY });
00236                 Point bottomLeft = CoordinateSystem.ToPlotCoordinates(new double[] { MinX, MaxY });
00237
00238                 double minX = Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X,
      bottomRight.X));
00239                 double minY = Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y,
      bottomRight.Y));
00240                 double maxX = Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X,
      bottomRight.X));
00241                 double maxY = Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y,
      bottomRight.Y));
00242
00243                 target.Save();
00244                 target.SetClippingPath(minX, minY, maxX - minX, maxY - minY);
00245
00246                 target.StrokePath(path, PresentationAttributes.Stroke,
      PresentationAttributes.LineWidth, PresentationAttributes.LineCap, PresentationAttributes.LineJoin,
      PresentationAttributes.LineDash, Tag);
00247                 target.Restore();
00248             }
00249
00250         }
00251     }
00252
00253 /// <summary>
00254 /// A plot element that draws an exponential trendline with equation <c>y = b * Exp(a * x)</c>.
00255 /// </summary>
00256     public class ExponentialTrendLine :  IPlotElement
00257     {
00258 /// <summary>
00259 /// The slope of the trendline (a).
00260 /// </summary>
00261         public double Slope { get; set; }
00262
00263 /// <summary>
00264 /// The intercept of the trendline (b).
00265 /// </summary>
00266         public double Intercept { get; set; }
00267
00268 /// <summary>
00269 /// The minimum X value for which the trendline is plotted.
00270 /// </summary>
00271         public double MinX { get; set; }
00272
00273 /// <summary>
00274 /// The minimum Y value for which the trendline is plotted.
00275 /// </summary>
00276         public double MinY { get; set; }
00277
00278 /// <summary>
00279 /// The maximum X value for which the trendline is plotted.
00280 /// </summary>
00281         public double MaxX { get; set; }
00282
00283 /// <summary>
00284 /// The maximum Y value for which the trendline is plotted.
```

```
00285 /// </summary>
00286         public double MaxY { get; set; }
00287 
00288 /// <summary>
00289 /// Presentation attributes for the trendline.
00290 /// </summary>
00291         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180,
      180, 180) };
00292 
00293 /// <summary>
00294 /// A tag to identify the trendline in the plot.
00295 /// </summary>
00296         public string Tag { get; set; }
00297 
00298 /// <summary>
00299 /// The coordinate system used to transform the points from data space to plot space.
00300 /// </summary>
00301         public IContinuousCoordinateSystem CoordinateSystem { get; set; }
00302         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00303 
00304 /// <summary>
00305 /// Create a new <see cref="ExponentialTrendLine"/> instance, specifying the equation parameters.
00306 /// </summary>
00307 /// <param name="slope">The slope of the trendline (a).</param>
00308 /// <param name="intercept">The intercept of the trendline (b).</param>
00309 /// <param name="minX">The minimum X value for which the trendline is plotted.</param>
00310 /// <param name="minY">The minimum Y value for which the trendline is plotted.</param>
00311 /// <param name="maxX">The maximum X value for which the trendline is plotted.</param>
00312 /// <param name="maxY">The maximum Y value for which the trendline is plotted.</param>
00313 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00314         public ExponentialTrendLine(double slope, double intercept, double minX, double minY, double
      maxX, double maxY, IContinuousCoordinateSystem coordinateSystem)
00315         {
00316             Slope = slope;
00317             Intercept = intercept;
00318             MinX = minX;
00319             MinY = minY;
00320             MaxX = maxX;
00321             MaxY = maxY;
00322             CoordinateSystem = coordinateSystem;
00323         }
00324 
00325 /// <summary>
00326 /// Create a new <see cref="ExponentialTrendLine"/> instance, determining the equation parameters by
      running a regression.
00327 /// </summary>
00328 /// <param name="data">The data that will be used to determine the equation parameters.</param>
00329 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00330 /// <param name="fixedIntercept">If this is <see langword="null"/>, the intercept (b) is determined
      during the regression; otherwise, it is fixed to the specified value.</param>
00331         public ExponentialTrendLine(IReadOnlyList<IReadOnlyList<double>> data,
      IContinuousCoordinateSystem coordinateSystem, double?  fixedIntercept = null)
00332         {
00333             if (fixedIntercept <= 0)
00334             {
00335                 throw new ArgumentOutOfRangeException(nameof(fixedIntercept), fixedIntercept, "The
      intercept must be greater than 0!");
00336             }
00337 
00338             MinX = double.MaxValue;
00339             MinY = double.MaxValue;
00340             MaxX = double.MinValue;
00341             MaxY = double.MinValue;
00342 
00343             double[] x = new double[data.Count];
00344             double[] y = new double[data.Count];
00345 
00346             double shift = 0;
00347 
00348             if (fixedIntercept != null)
00349             {
00350                 shift = Math.Log(fixedIntercept.Value);
00351             }
00352 
00353             for (int i = 0; i < data.Count; i++)
00354             {
00355                 MinX = Math.Min(MinX, data[i][0]);
00356                 MinY = Math.Min(MinY, data[i][1]);
00357                 MaxX = Math.Max(MaxX, data[i][0]);
00358                 MaxY = Math.Max(MaxY, data[i][1]);
00359 
00360                 x[i] = data[i][0];
00361                 y[i] = Math.Log(data[i][1]) - shift;
00362             }
```

```
00363
00364             if (fixedIntercept == null)
00365             {
00366                 (double k, double b) = MathNet.Numerics.LinearRegression.SimpleRegression.Fit(x, y);
00367                 Intercept = Math.Exp(k);
00368                 Slope = b;
00369             }
00370             else
00371             {
00372                 Slope = MathNet.Numerics.LinearRegression.SimpleRegression.FitThroughOrigin(x, y);
00373                 Intercept = fixedIntercept.Value;
00374             }
00375
00376             this.CoordinateSystem = coordinateSystem;
00377         }
00378
00379 /// <summary>
00380 /// Create a new <see cref="ExponentialTrendLine"/> instance, determining the equation parameters by running a regression.
00381 /// </summary>
00382 /// <param name="data">The data that will be used to determine the equation parameters.</param>
00383 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space to plot space.</param>
00384 /// <param name="fixedIntercept">If this is <see langword="null"/>, the intercept (b) is determined during the regression; otherwise, it is fixed to the specified value.</param>
00385         public ExponentialTrendLine(IReadOnlyList<(double, double)> data, IContinuousCoordinateSystem coordinateSystem, double?  fixedIntercept = null) :  this((from el in data select new double[] { el.Item1, el.Item2 }).ToArray(), coordinateSystem, fixedIntercept) { }
00386
00387 /// <inheritdoc/>
00388         public void Plot(Graphics target)
00389         {
00390             double y0 = Math.Exp(Slope * MinX) * Intercept;
00391             double x0 = (Math.Log(MinY) - Math.Log(Intercept)) / Slope;
00392
00393             double y1 = Math.Exp(Slope * MaxX) * Intercept;
00394             double x1 = (Math.Log(MaxY) - Math.Log(Intercept)) / Slope;
00395
00396             HashSet<(double, double)> points = new HashSet<(double, double)>();
00397
00398             if (y0 >= MinY && y0 <= MaxY)
00399             {
00400                 points.Add((MinX, y0));
00401             }
00402
00403             if (x0 >= MinX && x0 <= MaxX)
00404             {
00405                 points.Add((x0, MinY));
00406             }
00407
00408             if (y1 >= MinY && y1 <= MaxY)
00409             {
00410                 points.Add((MaxX, y1));
00411             }
00412
00413             if (x1 >= MinX && x1 <= MaxX)
00414             {
00415                 points.Add((x1, MaxY));
00416             }
00417
00418             if (points.Count == 2)
00419             {
00420                 double[][] uniquePoints = new double[points.Count][];
00421
00422                 int index = 0;
00423
00424                 foreach ((double x, double y) in points)
00425                 {
00426                     uniquePoints[index] = new double[] { x, y };
00427                     index++;
00428                 }
00429
00430                 double startX = Math.Min(uniquePoints[0][0], uniquePoints[1][0]);
00431                 double endX = Math.Max(uniquePoints[0][0], uniquePoints[1][0]);
00432
00433                 int count = (int)Math.Ceiling(Math.Max((endX - startX) / CoordinateSystem.Resolution[0], Math.Abs(uniquePoints[1][1] - uniquePoints[0][1]) / CoordinateSystem.Resolution[1]));
00434
00435                 GraphicsPath path = new GraphicsPath();
00436
00437                 for (int i = 0; i <= count; i++)
00438                 {
00439                     double[] pt = new double[] { startX + (endX - startX) / count * i, Math.Exp(Slope * (startX + (endX - startX) / count * i)) * Intercept };
00440
00441                     if (i == 0)
```

```
00442                     {
00443                         path.MoveTo(CoordinateSystem.ToPlotCoordinates(pt));
00444                     }
00445                     else
00446                     {
00447                         path.LineTo(CoordinateSystem.ToPlotCoordinates(pt));
00448                     }
00449                 }
00450 
00451                 Point topLeft = CoordinateSystem.ToPlotCoordinates(new double[] { MinX, MinY });
00452                 Point topRight = CoordinateSystem.ToPlotCoordinates(new double[] { MaxX, MinY });
00453                 Point bottomRight = CoordinateSystem.ToPlotCoordinates(new double[] { MaxX, MaxY });
00454                 Point bottomLeft = CoordinateSystem.ToPlotCoordinates(new double[] { MinX, MaxY });
00455 
00456                 double minX = Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X,
      bottomRight.X));
00457                 double minY = Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y,
      bottomRight.Y));
00458                 double maxX = Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X,
      bottomRight.X));
00459                 double maxY = Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y,
      bottomRight.Y));
00460 
00461                 target.Save();
00462                 target.SetClippingPath(minX, minY, maxX - minX, maxY - minY);
00463 
00464                 target.StrokePath(path, PresentationAttributes.Stroke,
      PresentationAttributes.LineWidth, PresentationAttributes.LineCap, PresentationAttributes.LineJoin,
      PresentationAttributes.LineDash, Tag);
00465                 target.Restore();
00466             }
00467 
00468         }
00469     }
00470 
00471 /// <summary>
00472 /// A plot element that draws a logarithmic trendline with equation <c>y = a * Ln(x) + b</c>.
00473 /// </summary>
00474     public class LogarithmicTrendLine :  IPlotElement
00475     {
00476 /// <summary>
00477 /// The slope of the trendline (a).
00478 /// </summary>
00479         public double Slope { get; set; }
00480 
00481 /// <summary>
00482 /// The intercept of the trendline (b).
00483 /// </summary>
00484         public double Intercept { get; set; }
00485 
00486 /// <summary>
00487 /// The minimum X value for which the trendline is plotted.
00488 /// </summary>
00489         public double MinX { get; set; }
00490 
00491 /// <summary>
00492 /// The minimum Y value for which the trendline is plotted.
00493 /// </summary>
00494         public double MinY { get; set; }
00495 
00496 /// <summary>
00497 /// The maximum X value for which the trendline is plotted.
00498 /// </summary>
00499         public double MaxX { get; set; }
00500 
00501 /// <summary>
00502 /// The maximum Y value for which the trendline is plotted.
00503 /// </summary>
00504         public double MaxY { get; set; }
00505 
00506 /// <summary>
00507 /// Presentation attributes for the trendline.
00508 /// </summary>
00509         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180,
      180, 180) };
00510 
00511 /// <summary>
00512 /// A tag to identify the trendline in the plot.
00513 /// </summary>
00514         public string Tag { get; set; }
00515 
00516 /// <summary>
00517 /// The coordinate system used to transform the points from data space to plot space.
00518 /// </summary>
00519         public IContinuousCoordinateSystem CoordinateSystem { get; set; }
00520         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
```

```
00521
00522 /// <summary>
00523 /// Create a new <see cref="LogarithmicTrendLine"/> instance, specifying the equation parameters.
00524 /// </summary>
00525 /// <param name="slope">The slope of the trendline (a).</param>
00526 /// <param name="intercept">The intercept of the trendline (b).</param>
00527 /// <param name="minX">The minimum X value for which the trendline is plotted.</param>
00528 /// <param name="minY">The minimum Y value for which the trendline is plotted.</param>
00529 /// <param name="maxX">The maximum X value for which the trendline is plotted.</param>
00530 /// <param name="maxY">The maximum Y value for which the trendline is plotted.</param>
00531 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00532         public LogarithmicTrendLine(double slope, double intercept, double minX, double minY, double
      maxX, double maxY, IContinuousCoordinateSystem coordinateSystem)
00533         {
00534             Slope = slope;
00535             Intercept = intercept;
00536             MinX = minX;
00537             MinY = minY;
00538             MaxX = maxX;
00539             MaxY = maxY;
00540             CoordinateSystem = coordinateSystem;
00541         }
00542
00543 /// <summary>
00544 /// Create a new <see cref="LogarithmicTrendLine"/> instance, determining the equation parameters by
      running a regression.
00545 /// </summary>
00546 /// <param name="data">The data that will be used to determine the equation parameters.</param>
00547 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00548 /// <param name="fixedIntercept">If this is <see langword="null"/>, the intercept (b) is determined
      during the regression; otherwise, it is fixed to the specified value.</param>
00549         public LogarithmicTrendLine(IReadOnlyList<IReadOnlyList<double>> data,
      IContinuousCoordinateSystem coordinateSystem, double?  fixedIntercept = null)
00550         {
00551             MinX = double.MaxValue;
00552             MinY = double.MaxValue;
00553             MaxX = double.MinValue;
00554             MaxY = double.MinValue;
00555
00556             double[] x = new double[data.Count];
00557             double[] y = new double[data.Count];
00558
00559             double shift = 0;
00560
00561             if (fixedIntercept != null)
00562             {
00563                 shift = fixedIntercept.Value;
00564             }
00565
00566             for (int i = 0; i < data.Count; i++)
00567             {
00568                 MinX = Math.Min(MinX, data[i][0]);
00569                 MinY = Math.Min(MinY, data[i][1]);
00570                 MaxX = Math.Max(MaxX, data[i][0]);
00571                 MaxY = Math.Max(MaxY, data[i][1]);
00572
00573                 x[i] = Math.Log(data[i][0]);
00574                 y[i] = data[i][1] - shift;
00575             }
00576
00577             if (fixedIntercept == null)
00578             {
00579                 (Intercept, Slope) = MathNet.Numerics.LinearRegression.SimpleRegression.Fit(x, y);
00580             }
00581             else
00582             {
00583                 Slope = MathNet.Numerics.LinearRegression.SimpleRegression.FitThroughOrigin(x, y);
00584                 Intercept = fixedIntercept.Value;
00585             }
00586
00587             this.CoordinateSystem = coordinateSystem;
00588         }
00589
00590 /// <summary>
00591 /// Create a new <see cref="LogarithmicTrendLine"/> instance, determining the equation parameters by
      running a regression.
00592 /// </summary>
00593 /// <param name="data">The data that will be used to determine the equation parameters.</param>
00594 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00595 /// <param name="fixedIntercept">If this is <see langword="null"/>, the intercept (b) is determined
      during the regression; otherwise, it is fixed to the specified value.</param>
00596         public LogarithmicTrendLine(IReadOnlyList<(double, double)> data, IContinuousCoordinateSystem
      coordinateSystem, double?  fixedIntercept = null) :  this((from el in data select new double[] {
      el.Item1, el.Item2 }).ToArray(), coordinateSystem, fixedIntercept) { }
```

```
00597
00598 /// <inheritdoc/>
00599         public void Plot(Graphics target)
00600         {
00601             double y0 = Slope * Math.Log(MinX) + Intercept;
00602             double x0 = Math.Exp((MinY - Intercept) / Slope);
00603
00604             double y1 = Slope * Math.Log(MaxX) + Intercept;
00605             double x1 = Math.Exp((MaxY - Intercept) / Slope);
00606
00607             HashSet<(double, double)> points = new HashSet<(double, double)>();
00608
00609             if (y0 >= MinY && y0 <= MaxY)
00610             {
00611                 points.Add((MinX, y0));
00612             }
00613
00614             if (x0 >= MinX && x0 <= MaxX)
00615             {
00616                 points.Add((x0, MinY));
00617             }
00618
00619             if (y1 >= MinY && y1 <= MaxY)
00620             {
00621                 points.Add((MaxX, y1));
00622             }
00623
00624             if (x1 >= MinX && x1 <= MaxX)
00625             {
00626                 points.Add((x1, MaxY));
00627             }
00628
00629             if (points.Count == 2)
00630             {
00631                 double[][] uniquePoints = new double[points.Count][];
00632
00633                 int index = 0;
00634
00635                 foreach ((double x, double y) in points)
00636                 {
00637                     uniquePoints[index] = new double[] { x, y };
00638                     index++;
00639                 }
00640
00641                 double startX = Math.Min(uniquePoints[0][0], uniquePoints[1][0]);
00642                 double endX = Math.Max(uniquePoints[0][0], uniquePoints[1][0]);
00643
00644                 int count = (int)Math.Ceiling(Math.Max((endX - startX) /
    CoordinateSystem.Resolution[0], Math.Abs(uniquePoints[1][1] - uniquePoints[0][1]) /
    CoordinateSystem.Resolution[1]));
00645
00646                 GraphicsPath path = new GraphicsPath();
00647
00648                 for (int i = 0; i <= count; i++)
00649                 {
00650                     double[] pt = new double[] { startX + (endX - startX) / count * i, Slope *
    Math.Log(startX + (endX - startX) / count * i) + Intercept };
00651
00652                     if (i == 0)
00653                     {
00654                         path.MoveTo(CoordinateSystem.ToPlotCoordinates(pt));
00655                     }
00656                     else
00657                     {
00658                         path.LineTo(CoordinateSystem.ToPlotCoordinates(pt));
00659                     }
00660                 }
00661
00662                 Point topLeft = CoordinateSystem.ToPlotCoordinates(new double[] { MinX, MinY });
00663                 Point topRight = CoordinateSystem.ToPlotCoordinates(new double[] { MaxX, MinY });
00664                 Point bottomRight = CoordinateSystem.ToPlotCoordinates(new double[] { MaxX, MaxY });
00665                 Point bottomLeft = CoordinateSystem.ToPlotCoordinates(new double[] { MinX, MaxY });
00666
00667                 double minX = Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X,
    bottomRight.X));
00668                 double minY = Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y,
    bottomRight.Y));
00669                 double maxX = Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X,
    bottomRight.X));
00670                 double maxY = Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y,
    bottomRight.Y));
00671
00672                 target.Save();
00673                 target.SetClippingPath(minX, minY, maxX - minX, maxY - minY);
00674
00675                 target.StrokePath(path, PresentationAttributes.Stroke,
    PresentationAttributes.LineWidth, PresentationAttributes.LineCap, PresentationAttributes.LineJoin,
```

```
      PresentationAttributes.LineDash, Tag);
00676                 target.Restore();
00677             }
00678 
00679         }
00680     }
00681 
00682 /// <summary>
00683 /// A plot element that draws a polynomial trendline with equation <c>y = a0 + a1 * x + a2 * x^2 + ...
      + aN * x^N</c>.
00684 /// </summary>
00685     public class PolynomialTrendLine :  IPlotElement
00686     {
00687 /// <summary>
00688 /// The coefficients (<c>a0 ...  aN</c>).
00689 /// </summary>
00690         public double[] Coefficients { get; set; }
00691 /// <summary>
00692 /// The minimum X value for which the trendline is plotted.
00693 /// </summary>
00694         public double MinX { get; set; }
00695 
00696 /// <summary>
00697 /// The minimum Y value for which the trendline is plotted.
00698 /// </summary>
00699         public double MinY { get; set; }
00700 
00701 /// <summary>
00702 /// The maximum X value for which the trendline is plotted.
00703 /// </summary>
00704         public double MaxX { get; set; }
00705 
00706 /// <summary>
00707 /// The maximum Y value for which the trendline is plotted.
00708 /// </summary>
00709         public double MaxY { get; set; }
00710 
00711 /// <summary>
00712 /// Presentation attributes for the trendline.
00713 /// </summary>
00714         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180,
      180, 180) };
00715 
00716 /// <summary>
00717 /// A tag to identify the trendline in the plot.
00718 /// </summary>
00719         public string Tag { get; set; }
00720 
00721 /// <summary>
00722 /// The coordinate system used to transform the points from data space to plot space.
00723 /// </summary>
00724         public IContinuousCoordinateSystem CoordinateSystem { get; set; }
00725         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00726 
00727 /// <summary>
00728 /// Create a new <see cref="LinearTrendLine"/> instance, specifying the coefficients.
00729 /// </summary>
00730 /// <param name="coefficients">The coefficients (<c>a0 ...  aN</c>).</param>
00731 /// <param name="minX">The minimum X value for which the trendline is plotted.</param>
00732 /// <param name="minY">The minimum Y value for which the trendline is plotted.</param>
00733 /// <param name="maxX">The maximum X value for which the trendline is plotted.</param>
00734 /// <param name="maxY">The maximum Y value for which the trendline is plotted.</param>
00735 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00736         public PolynomialTrendLine(double[] coefficients, double minX, double minY, double maxX,
      double maxY, IContinuousCoordinateSystem coordinateSystem)
00737         {
00738             Coefficients = coefficients;
00739             MinX = minX;
00740             MinY = minY;
00741             MaxX = maxX;
00742             MaxY = maxY;
00743             CoordinateSystem = coordinateSystem;
00744         }
00745 
00746         private static double[] FitPolynomialWithFixedIntercept(double[] x, double[] y, int order,
      double fixedIntercept)
00747         {
00748             Matrix<double> v = new DenseMatrix(x.Length, order);
00749 
00750             for (int i = 0; i < v.RowCount; i++)
00751             {
00752                 for (int j = 0; j < order; j++)
00753                 {
00754                     v[i, j] = Math.Pow(x[i], j + 1);
00755                 }
```

```
00756             }
00757
00758             double[] coeffs = MultipleRegression.QR(v, Vector<double>.Build.Dense(y)).ToArray();
00759             double[] tbr = new double[coeffs.Length + 1];
00760             tbr[0] = fixedIntercept;
00761             for (int i = 0; i < coeffs.Length; i++)
00762             {
00763                 tbr[i + 1] = coeffs[i];
00764             }
00765
00766             return tbr;
00767         }
00768
00769 /// <summary>
00770 /// Create a new <see cref="PolynomialTrendLine"/> instance, determining the coefficients by running a
      regression.
00771 /// </summary>
00772 /// <param name="data">The data that will be used to determine the coefficients.</param>
00773 /// <param name="order">The order of the polynomial (<c>N</c>).  This must be ≥ 2.</param>
00774 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00775 /// <param name="fixedIntercept">If this is <see langword="null"/>, the intercept (a0) is determined
      during the regression; otherwise, it is fixed to the specified value.</param>
00776 /// <exception cref="ArgumentOutOfRangeException">Thrown if the <paramref name="order"/> is &lt;
      2.</exception>
00777         public PolynomialTrendLine(IReadOnlyList<IReadOnlyList<double» data, int order,
      IContinuousCoordinateSystem coordinateSystem,  double?  fixedIntercept = null)
00778         {
00779             if (order < 2)
00780             {
00781                 throw new ArgumentOutOfRangeException(nameof(order), order, "The polynomial order must
      be 2 or greater!");
00782             }
00783
00784             MinX = double.MaxValue;
00785             MinY = double.MaxValue;
00786             MaxX = double.MinValue;
00787             MaxY = double.MinValue;
00788
00789             double[] x = new double[data.Count];
00790             double[] y = new double[data.Count];
00791
00792             double shift = 0;
00793
00794             if (fixedIntercept != null)
00795             {
00796                 shift = fixedIntercept.Value;
00797             }
00798
00799             for (int i = 0; i < data.Count; i++)
00800             {
00801                 MinX = Math.Min(MinX, data[i][0]);
00802                 MinY = Math.Min(MinY, data[i][1]);
00803                 MaxX = Math.Max(MaxX, data[i][0]);
00804                 MaxY = Math.Max(MaxY, data[i][1]);
00805
00806                 x[i] = data[i][0];
00807                 y[i] = data[i][1] - shift;
00808             }
00809
00810             if (fixedIntercept == null)
00811             {
00812                 Coefficients = MathNet.Numerics.Fit.Polynomial(x, y, order);
00813             }
00814             else
00815             {
00816                 Coefficients = FitPolynomialWithFixedIntercept(x, y, order, fixedIntercept.Value);
00817             }
00818
00819             this.CoordinateSystem = coordinateSystem;
00820         }
00821
00822 /// <summary>
00823 /// Create a new <see cref="PolynomialTrendLine"/> instance, determining the coefficients by running a
      regression.
00824 /// </summary>
00825 /// <param name="data">The data that will be used to determine the coefficients.</param>
00826 /// <param name="order">The order of the polynomial (<c>N</c>).  This must be ≥ 2.</param>
00827 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00828 /// <param name="fixedIntercept">If this is <see langword="null"/>, the intercept (a0) is determined
      during the regression; otherwise, it is fixed to the specified value.</param>
00829 /// <exception cref="ArgumentOutOfRangeException">Thrown if the <paramref name="order"/> is &lt;
      2.</exception>
00830         public PolynomialTrendLine(IReadOnlyList<(double, double)> data, int order,
      IContinuousCoordinateSystem coordinateSystem,  double?  fixedIntercept = null) :  this((from el in
      data select new double[] { el.Item1, el.Item2 }).ToArray(), order, coordinateSystem, fixedIntercept) {
```

```
    }
00831
00832 /// <inheritdoc/>
00833         public void Plot(Graphics target)
00834         {
00835             double startX = Math.Min(MinX, MaxX);
00836             double endX = Math.Max(MinX, MaxX);
00837
00838             int count = (int)Math.Ceiling((endX - startX) / CoordinateSystem.Resolution[0]);
00839
00840             GraphicsPath path = new GraphicsPath();
00841
00842             for (int i = 0; i <= count; i++)
00843             {
00844                 double[] pt = new double[] { startX + (endX - startX) / count * i,
    Polynomial.Evaluate(startX + (endX - startX) / count * i, Coefficients) };
00845
00846                 if (i == 0)
00847                 {
00848                     path.MoveTo(CoordinateSystem.ToPlotCoordinates(pt));
00849                 }
00850                 else
00851                 {
00852                     path.LineTo(CoordinateSystem.ToPlotCoordinates(pt));
00853                 }
00854             }
00855
00856             Point topLeft = CoordinateSystem.ToPlotCoordinates(new double[] { MinX, MinY });
00857             Point topRight = CoordinateSystem.ToPlotCoordinates(new double[] { MaxX, MinY });
00858             Point bottomRight = CoordinateSystem.ToPlotCoordinates(new double[] { MaxX, MaxY });
00859             Point bottomLeft = CoordinateSystem.ToPlotCoordinates(new double[] { MinX, MaxY });
00860
00861             double minX = Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X,
    bottomRight.X));
00862             double minY = Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y,
    bottomRight.Y));
00863             double maxX = Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X,
    bottomRight.X));
00864             double maxY = Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y,
    bottomRight.Y));
00865
00866             target.Save();
00867             target.SetClippingPath(minX, minY, maxX - minX, maxY - minY);
00868
00869             target.StrokePath(path, PresentationAttributes.Stroke, PresentationAttributes.LineWidth,
    PresentationAttributes.LineCap, PresentationAttributes.LineJoin, PresentationAttributes.LineDash,
    Tag);
00870             target.Restore();
00871         }
00872     }
00873
00874 /// <summary>
00875 /// A plot element that draws a power law trendline with equation <c>y = b * x^a</c>.
00876 /// </summary>
00877     public class PowerLawTrendLine :  IPlotElement
00878     {
00879 /// <summary>
00880 /// The slope of the trendline (a).
00881 /// </summary>
00882         public double Slope { get; set; }
00883
00884 /// <summary>
00885 /// The intercept of the trendline (b).
00886 /// </summary>
00887         public double Intercept { get; set; }
00888
00889 /// <summary>
00890 /// The minimum X value for which the trendline is plotted.
00891 /// </summary>
00892         public double MinX { get; set; }
00893
00894 /// <summary>
00895 /// The minimum Y value for which the trendline is plotted.
00896 /// </summary>
00897         public double MinY { get; set; }
00898
00899 /// <summary>
00900 /// The maximum X value for which the trendline is plotted.
00901 /// </summary>
00902         public double MaxX { get; set; }
00903
00904 /// <summary>
00905 /// The maximum Y value for which the trendline is plotted.
00906 /// </summary>
00907         public double MaxY { get; set; }
00908
00909 /// <summary>
```

```
00910 /// Presentation attributes for the trendline.
00911 /// </summary>
00912         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180,
      180, 180) };
00913
00914 /// <summary>
00915 /// A tag to identify the trendline in the plot.
00916 /// </summary>
00917         public string Tag { get; set; }
00918
00919 /// <summary>
00920 /// The coordinate system used to transform the points from data space to plot space.
00921 /// </summary>
00922         public IContinuousCoordinateSystem CoordinateSystem { get; set; }
00923         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
00924
00925 /// <summary>
00926 /// Create a new <see cref="PowerLawTrendLine"/> instance, specifying the equation parameters.
00927 /// </summary>
00928 /// <param name="slope">The slope of the trendline (a).</param>
00929 /// <param name="intercept">The intercept of the trendline (b).</param>
00930 /// <param name="minX">The minimum X value for which the trendline is plotted.</param>
00931 /// <param name="minY">The minimum Y value for which the trendline is plotted.</param>
00932 /// <param name="maxX">The maximum X value for which the trendline is plotted.</param>
00933 /// <param name="maxY">The maximum Y value for which the trendline is plotted.</param>
00934 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00935         public PowerLawTrendLine(double slope, double intercept, double minX, double minY, double
      maxX, double maxY, IContinuousCoordinateSystem coordinateSystem)
00936         {
00937             Slope = slope;
00938             Intercept = intercept;
00939             MinX = minX;
00940             MinY = minY;
00941             MaxX = maxX;
00942             MaxY = maxY;
00943             CoordinateSystem = coordinateSystem;
00944         }
00945
00946 /// <summary>
00947 /// Create a new <see cref="PowerLawTrendLine"/> instance, determining the equation parameters by
      running a regression.
00948 /// </summary>
00949 /// <param name="data">The data that will be used to determine the equation parameters.</param>
00950 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00951         public PowerLawTrendLine(IReadOnlyList<IReadOnlyList<double» data, IContinuousCoordinateSystem
      coordinateSystem)
00952         {
00953             MinX = double.MaxValue;
00954             MinY = double.MaxValue;
00955             MaxX = double.MinValue;
00956             MaxY = double.MinValue;
00957
00958             double[] x = new double[data.Count];
00959             double[] y = new double[data.Count];
00960
00961             for (int i = 0; i < data.Count; i++)
00962             {
00963                 MinX = Math.Min(MinX, data[i][0]);
00964                 MinY = Math.Min(MinY, data[i][1]);
00965                 MaxX = Math.Max(MaxX, data[i][0]);
00966                 MaxY = Math.Max(MaxY, data[i][1]);
00967
00968                 x[i] = Math.Log(data[i][0]);
00969                 y[i] = Math.Log(data[i][1]);
00970             }
00971
00972             (double intercept, double slope) =
      MathNet.Numerics.LinearRegression.SimpleRegression.Fit(x, y);
00973
00974             Slope = slope;
00975             Intercept = Math.Exp(intercept);
00976
00977             this.CoordinateSystem = coordinateSystem;
00978         }
00979
00980 /// <summary>
00981 /// Create a new <see cref="PowerLawTrendLine"/> instance, determining the equation parameters by
      running a regression.
00982 /// </summary>
00983 /// <param name="data">The data that will be used to determine the equation parameters.</param>
00984 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
00985         public PowerLawTrendLine(IReadOnlyList<(double, double)> data, IContinuousCoordinateSystem
      coordinateSystem) :  this((from el in data select new double[] { el.Item1, el.Item2 }).ToArray(),
```

```
coordinateSystem) { }
00986
00987 /// <inheritdoc/>
00988         public void Plot(Graphics target)
00989         {
00990             double y0 = Intercept * Math.Pow(MinX, Slope);
00991             double x0 = Math.Pow(MinY / Intercept, 1 / Slope);
00992
00993             double y1 = Intercept * Math.Pow(MaxX, Slope);
00994             double x1 = Math.Pow(MaxY / Intercept, 1 / Slope);
00995
00996             HashSet<(double, double)> points = new HashSet<(double, double)>();
00997
00998             if (y0 >= MinY && y0 <= MaxY)
00999             {
01000                 points.Add((MinX, y0));
01001             }
01002
01003             if (x0 >= MinX && x0 <= MaxX)
01004             {
01005                 points.Add((x0, MinY));
01006             }
01007
01008             if (y1 >= MinY && y1 <= MaxY)
01009             {
01010                 points.Add((MaxX, y1));
01011             }
01012
01013             if (x1 >= MinX && x1 <= MaxX)
01014             {
01015                 points.Add((x1, MaxY));
01016             }
01017
01018             if (points.Count == 2)
01019             {
01020                 double[][] uniquePoints = new double[points.Count][];
01021
01022                 int index = 0;
01023
01024                 foreach ((double x, double y) in points)
01025                 {
01026                     uniquePoints[index] = new double[] { x, y };
01027                     index++;
01028                 }
01029
01030                 double startX = Math.Min(uniquePoints[0][0], uniquePoints[1][0]);
01031                 double endX = Math.Max(uniquePoints[0][0], uniquePoints[1][0]);
01032
01033                 int count = (int)Math.Ceiling(Math.Max((endX - startX) /
    CoordinateSystem.Resolution[0], Math.Abs(uniquePoints[1][1] - uniquePoints[0][1]) /
    CoordinateSystem.Resolution[1]));
01034
01035                 GraphicsPath path = new GraphicsPath();
01036
01037                 for (int i = 0; i <= count; i++)
01038                 {
01039                     double[] pt = new double[] { startX + (endX - startX) / count * i, Intercept *
    Math.Pow(startX + (endX - startX) / count * i, Slope) };
01040
01041                     if (i == 0)
01042                     {
01043                         path.MoveTo(CoordinateSystem.ToPlotCoordinates(pt));
01044                     }
01045                     else
01046                     {
01047                         path.LineTo(CoordinateSystem.ToPlotCoordinates(pt));
01048                     }
01049                 }
01050
01051                 Point topLeft = CoordinateSystem.ToPlotCoordinates(new double[] { MinX, MinY });
01052                 Point topRight = CoordinateSystem.ToPlotCoordinates(new double[] { MaxX, MinY });
01053                 Point bottomRight = CoordinateSystem.ToPlotCoordinates(new double[] { MaxX, MaxY });
01054                 Point bottomLeft = CoordinateSystem.ToPlotCoordinates(new double[] { MinX, MaxY });
01055
01056                 double minX = Math.Min(Math.Min(topLeft.X, topRight.X), Math.Min(bottomLeft.X,
    bottomRight.X));
01057                 double minY = Math.Min(Math.Min(topLeft.Y, topRight.Y), Math.Min(bottomLeft.Y,
    bottomRight.Y));
01058                 double maxX = Math.Max(Math.Max(topLeft.X, topRight.X), Math.Max(bottomLeft.X,
    bottomRight.X));
01059                 double maxY = Math.Max(Math.Max(topLeft.Y, topRight.Y), Math.Max(bottomLeft.Y,
    bottomRight.Y));
01060
01061                 target.Save();
01062                 target.SetClippingPath(minX, minY, maxX - minX, maxY - minY);
01063
01064                 target.StrokePath(path, PresentationAttributes.Stroke,
```

```
      PresentationAttributes.LineWidth, PresentationAttributes.LineCap, PresentationAttributes.LineJoin,
      PresentationAttributes.LineDash, Tag);
01065                 target.Restore();
01066             }
01067
01068         }
01069     }
01070
01071 /// <summary>
01072 /// A plot element that draws a moving average trendline.
01073 /// </summary>
01074     public class MovingAverageTrendLine :  IPlotElement
01075     {
01076 /// <summary>
01077 /// The data used to compute the moving average.  These must be sorted in a sensible way.
01078 /// </summary>
01079         public IReadOnlyList<IReadOnlyList<double» Data { get; set; }
01080
01081 /// <summary>
01082 /// The weight function.  This should accept two parameters:  an <see langword="int"/> representing
      the difference in
01083 /// index between the point being weighted and the "focus point", and a <see langword="double"/>[]
      representing the
01084 /// difference in coordinates between the two points.  It should return a <see langword="double"/>
      representing the
01085 /// weight (it does not need to be normalised).
01086 /// </summary>
01087         public Func<int, double[], double> Weight { get; set; } = (int i, double[] d) => 1;
01088
01089 /// <summary>
01090 /// The number of points that are averaged to obtain the value for each point.  This must be ≥ 0.
01091 /// </summary>
01092         public int Period { get; set; }
01093
01094 /// <summary>
01095 /// Presentation attributes for the trendline.
01096 /// </summary>
01097         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes() { LineDash = new LineDash(5, 5, 0), Stroke = Colour.FromRgb(180,
      180, 180) };
01098
01099 /// <summary>
01100 /// A tag to identify the trendline in the plot.
01101 /// </summary>
01102         public string Tag { get; set; }
01103
01104 /// <summary>
01105 /// The coordinate system used to transform the points from data space to plot space.
01106 /// </summary>
01107         public ICoordinateSystem<IReadOnlyList<double>> CoordinateSystem { get; set; }
01108         ICoordinateSystem IPlotElement.CoordinateSystem => CoordinateSystem;
01109
01110 /// <summary>
01111 /// Create a new <see cref="MovingAverageTrendLine"/> instance from the specified data.
01112 /// </summary>
01113 /// <param name="data">The data used to compute the moving average.  These must be sorted in a
      sensible way.</param>
01114 /// <param name="period">The number of points that are averaged to obtain the value for each point.
      This must be ≥ 0.</param>
01115 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
01116 /// <exception cref="ArgumentOutOfRangeException">Thrown if the <paramref name="period"/> is &lt;
      0.</exception>
01117         public MovingAverageTrendLine(IReadOnlyList<IReadOnlyList<double» data, int period,
      ICoordinateSystem<IReadOnlyList<double» coordinateSystem)
01118         {
01119             if (period < 0)
01120             {
01121                 throw new ArgumentOutOfRangeException(nameof(period), period, "The period of the
      moving average must be greater than or equal to zero!");
01122             }
01123
01124             this.Data = data;
01125             this.Period = period;
01126             CoordinateSystem = coordinateSystem;
01127         }
01128
01129 /// <summary>
01130 /// Create a new <see cref="MovingAverageTrendLine"/> instance from the specified data.
01131 /// </summary>
01132 /// <param name="data">The data used to compute the moving average.  These must be sorted in a
      sensible way.</param>
01133 /// <param name="period">The number of points that are averaged to obtain the value for each point.
      This must be ≥ 0.</param>
01134 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
      to plot space.</param>
01135 /// <exception cref="ArgumentOutOfRangeException">Thrown if the <paramref name="period"/> is &lt;
```

```
    0.</exception>
01136         public MovingAverageTrendLine(IReadOnlyList<(double, double)> data, int period,
    IContinuousCoordinateSystem coordinateSystem) :  this((from el in data select new double[] { el.Item1,
    el.Item2 }).ToArray(), period, coordinateSystem) { }
01137
01138 /// <inheritdoc/>
01139         public void Plot(Graphics target)
01140         {
01141             GraphicsPath path = new GraphicsPath();
01142
01143             for (int i = 0; i < Data.Count; i++)
01144             {
01145                 double count = 0;
01146
01147                 double[] point = new double[Data[i].Count];
01148
01149                 for (int k = -Period; k <= Period; k++)
01150                 {
01151                     if (i + k >= 0 && i + k < Data.Count)
01152                     {
01153                         double[] diff = new double[Data[i].Count];
01154
01155                         for (int j = 0; j < Data[i].Count; j++)
01156                         {
01157                             diff[j] = Data[i + k][j] - Data[i][j];
01158                         }
01159
01160                         double weighting = Weight(k, diff);
01161
01162                         count += weighting;
01163
01164                         for (int j = 0; j < point.Length; j++)
01165                         {
01166                             point[j] += Data[i + k][j] * weighting;
01167                         }
01168                     }
01169                 }
01170
01171                 for (int j = 0; j < point.Length; j++)
01172                 {
01173                     point[j] /= count;
01174                 }
01175
01176                 if (i == 0)
01177                 {
01178                     path.MoveTo(CoordinateSystem.ToPlotCoordinates(point));
01179                 }
01180                 else
01181                 {
01182                     path.LineTo(CoordinateSystem.ToPlotCoordinates(point));
01183                 }
01184             }
01185
01186             target.StrokePath(path, PresentationAttributes.Stroke, PresentationAttributes.LineWidth,
    PresentationAttributes.LineCap, PresentationAttributes.LineJoin, PresentationAttributes.LineDash,
    Tag);
01187
01188         }
01189     }
01190 }
```

8.57 Violin.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2023 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
```

```
00021
00022 namespace VectSharp.Plots
00023 {
00024 /// <summary>
00025 /// A plot element that draws a violin plot.
00026 /// </summary>
00027     public class Violin :  IPlotElement
00028     {
00029 /// <summary>
00030 /// The sides on which a violin can be drawn.
00031 /// </summary>
00032         public enum ViolinSide
00033         {
00034 /// <summary>
00035 /// Draw the violin only on the left side.
00036 /// </summary>
00037             Left,
00038
00039 /// <summary>
00040 /// Draw the violin only on the right side.
00041 /// </summary>
00042             Right,
00043
00044 /// <summary>
00045 /// Draw the violin on both left and right sides.
00046 /// </summary>
00047             Both
00048         }
00049
00050 /// <summary>
00051 /// The position of the origin of the violin (e.g., the 0 in data space coordinates).
00052 /// </summary>
00053         public IReadOnlyList<double> Position { get; set; }
00054
00055 /// <summary>
00056 /// The direction along which the violin is drawn, in data space coordinates.
00057 /// </summary>
00058         public IReadOnlyList<double> Direction { get; set; }
00059
00060 /// <summary>
00061 /// The width of the violin in data space coordinates.
00062 /// </summary>
00063         public double Width { get; set; } = 10;
00064
00065 /// <summary>
00066 /// Determines whether the violin is smoothed or not.
00067 /// </summary>
00068         public bool Smooth { get; set; } = true;
00069
00070 /// <summary>
00071 /// Determines on which side(s) the violin is drawn.
00072 /// </summary>
00073         public ViolinSide Sides { get; set; } = ViolinSide.Both;
00074
00075 /// <summary>
00076 /// The values whose distribution is displayed by the violin plot.
00077 /// </summary>
00078         public IReadOnlyList<double> Data { get; set; }
00079
00080 /// <summary>
00081 /// Maximum number of bins.
00082 /// </summary>
00083         public int MaxBins { get; set; } = int.MaxValue;
00084
00085 /// <summary>
00086 /// Presentation attributes for the violin plot.
00087 /// </summary>
00088         public PlotElementPresentationAttributes PresentationAttributes { get; set; } = new
      PlotElementPresentationAttributes() { Fill = Colours.White };
00089
00090 /// <summary>
00091 /// The coordinate system used to transform the points from data space to plot space.
00092 /// </summary>
00093         public ICoordinateSystem<IReadOnlyList<double>> CoordinateSystem { get; set; }
00094         ICoordinateSystem IPlotElement.CoordinateSystem => this.CoordinateSystem;
00095
00096 /// <summary>
00097 /// A tag to identify the violin in the plot.
00098 /// </summary>
00099         public string Tag { get; set; }
00100
00101 /// <summary>
00102 /// Create a new <see cref="Violin"/> instance.
00103 /// </summary>
00104 /// <param name="position">The position of the origin of the violin (e.g., the 0 in data space
      coordinates).</param>
00105 /// <param name="direction">The direction along which the violin is drawn, in data space
```

```
coordinates.</param>
00106 /// <param name="data">The values whose distribution is displayed by the violin plot.</param>
00107 /// <param name="coordinateSystem">The coordinate system used to transform the points from data space
to plot space.</param>
00108         public Violin(IReadOnlyList<double> position, IReadOnlyList<double> direction,
IReadOnlyList<double> data, ICoordinateSystem<IReadOnlyList<double» coordinateSystem)
00109         {
00110             this.Position = position;
00111             this.Direction = direction;
00112             this.Data = data;
00113             this.CoordinateSystem = coordinateSystem;
00114         }
00115
00116 /// <inheritdoc/>
00117         public void Plot(Graphics target)
00118         {
00119
00120             double min = double.MaxValue;
00121             double max = double.MinValue;
00122
00123             for (int i = 0; i < Data.Count; i++)
00124             {
00125                 min = Math.Min(min, Data[i]);
00126                 max = Math.Max(max, Data[i]);
00127             }
00128
00129             (double _, double _, double iqr) = Plots.Plot.IQR(Data);
00130             double h2 = 2 * iqr / Math.Pow(Data.Count, 1.0 / 3.0);
00131             int binCount = Math.Max(1, (int)Math.Ceiling((max - min) / h2));
00132
00133             binCount = Math.Min(binCount, MaxBins);
00134
00135             int[] bins = new int[binCount];
00136
00137             if (max > min)
00138             {
00139                 for (int i = 0; i < Data.Count; i++)
00140                 {
00141                     int index = (int)Math.Min(binCount - 1, Math.Floor((Data[i] - min) / (max - min) *
binCount));
00142
00143                     bins[index]++;
00144                 }
00145             }
00146             else
00147             {
00148                 bins[0] = Data.Count;
00149             }
00150
00151             double binMax = bins.Max();
00152
00153             double[] perpDirection = new double[] { Direction[1], -Direction[0] };
00154
00155             List<Point> points = new List<Point>();
00156             List<Point> pointsOpposite = new List<Point>();
00157
00158             Point topPoint = CoordinateSystem.ToPlotCoordinates(new double[] { Position[0] +
Direction[0] * min, Position[1] + Direction[1] * min });
00159             Point bottomPoint = CoordinateSystem.ToPlotCoordinates(new double[] { Position[0] +
Direction[0] * max, Position[1] + Direction[1] * max });
00160
00161             for (int i = 0; i < bins.Length; i++)
00162             {
00163                 double binStart = min + (max - min) / binCount * i;
00164                 double binEnd = min + (max - min) / binCount * (i + 1);
00165
00166                 double x = (binStart + binEnd) * 0.5;
00167
00168                 if (i == 0)
00169                 {
00170                     if (bins.Length > 1 && bins[i + 1] > bins[i])
00171                     {
00172                         points.Add(CoordinateSystem.ToPlotCoordinates(new double[] { Position[0] +
Direction[0] * binStart, Position[1] + Direction[1] * binStart }));
00173                         pointsOpposite.Add(CoordinateSystem.ToPlotCoordinates(new double[] {
Position[0] + Direction[0] * binStart, Position[1] + Direction[1] * binStart }));
00174                     }
00175                     else
00176                     {
00177                         points.Add(CoordinateSystem.ToPlotCoordinates(new double[] { Position[0] +
Direction[0] * binStart + perpDirection[0] * bins[i] / binMax * Width, Position[1] + Direction[1] *
binStart + perpDirection[1] * bins[i] / binMax * Width }));
00178                         pointsOpposite.Add(CoordinateSystem.ToPlotCoordinates(new double[] {
Position[0] + Direction[0] * binStart - perpDirection[0] * bins[i] / binMax * Width, Position[1] +
Direction[1] * binStart - perpDirection[1] * bins[i] / binMax * Width }));
00179                     }
00180                 }
```

```
00181
00182                 points.Add(CoordinateSystem.ToPlotCoordinates(new double[] { Position[0] +
      Direction[0] * x + perpDirection[0] * bins[i] / binMax * Width, Position[1] + Direction[1] * x +
      perpDirection[1] * bins[i] / binMax * Width }));
00183                 pointsOpposite.Add(CoordinateSystem.ToPlotCoordinates(new double[] { Position[0] +
      Direction[0] * x - perpDirection[0] * bins[i] / binMax * Width, Position[1] + Direction[1] * x -
      perpDirection[1] * bins[i] / binMax * Width }));
00184
00185                 if (i == bins.Length - 1)
00186                 {
00187                     if (bins.Length > 1 && bins[i - 1] > bins[i])
00188                     {
00189                         points.Add(CoordinateSystem.ToPlotCoordinates(new double[] { Position[0] +
      Direction[0] * binEnd, Position[1] + Direction[1] * binEnd }));
00190                         pointsOpposite.Add(CoordinateSystem.ToPlotCoordinates(new double[] {
      Position[0] + Direction[0] * binEnd, Position[1] + Direction[1] * binEnd }));
00191                     }
00192                     else
00193                     {
00194                         points.Add(CoordinateSystem.ToPlotCoordinates(new double[] { Position[0] +
      Direction[0] * binEnd + perpDirection[0] * bins[i] / binMax * Width, Position[1] + Direction[1] *
      binEnd + perpDirection[1] * bins[i] / binMax * Width }));
00195                         pointsOpposite.Add(CoordinateSystem.ToPlotCoordinates(new double[] {
      Position[0] + Direction[0] * binEnd - perpDirection[0] * bins[i] / binMax * Width, Position[1] +
      Direction[1] * binEnd - perpDirection[1] * bins[i] / binMax * Width }));
00196                     }
00197                 }
00198             }
00199
00200             pointsOpposite.Reverse();
00201
00202             GraphicsPath path = new GraphicsPath();
00203
00204             if (Sides == ViolinSide.Left || Sides == ViolinSide.Both)
00205             {
00206                 if (Smooth)
00207                 {
00208                     path.AddSmoothSpline(pointsOpposite.ToArray());
00209                 }
00210                 else
00211                 {
00212                     for (int i= 0; i < pointsOpposite.Count; i++)
00213                     {
00214                         if (i > 0)
00215                         {
00216                             path.LineTo(pointsOpposite[i]);
00217                         }
00218                         else
00219                         {
00220                             path.MoveTo(pointsOpposite[i]);
00221                         }
00222                     }
00223                 }
00224             }
00225             else
00226             {
00227                 path.MoveTo(bottomPoint).LineTo(topPoint);
00228             }
00229
00230             if (Sides == ViolinSide.Right || Sides == ViolinSide.Both)
00231             {
00232                 if (Smooth)
00233                 {
00234                     path.AddSmoothSpline(points.ToArray());
00235                 }
00236                 else
00237                 {
00238                     for (int i = 0; i < points.Count; i++)
00239                     {
00240                         path.LineTo(points[i]);
00241                     }
00242                 }
00243             }
00244             else
00245             {
00246                 path.LineTo(topPoint).LineTo(bottomPoint);
00247             }
00248
00249             path.Close();
00250
00251             string tag = Tag;
00252
00253             string strokeTag = Tag;
00254
00255             if (!string.IsNullOrEmpty(Tag) && target.UseUniqueTags)
00256             {
00257                 strokeTag += "@stroke";
```

```
00258             }
00259
00260
00261             if (PresentationAttributes.Fill != null)
00262             {
00263                 target.FillPath(path, PresentationAttributes.Fill, tag);
00264             }
00265
00266             if (PresentationAttributes.Stroke != null)
00267             {
00268                 target.StrokePath(path, PresentationAttributes.Stroke,
      PresentationAttributes.LineWidth, PresentationAttributes.LineCap, PresentationAttributes.LineJoin,
      PresentationAttributes.LineDash, strokeTag);
00269             }
00270         }
00271     }
00272 }
```

8.58 GradientBrushApplicator.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Buffers;
00020 using System.Numerics;
00021 using SixLabors.ImageSharp.Memory;
00022 using SixLabors.ImageSharp.PixelFormats;
00023 using SixLabors.ImageSharp.Drawing.Processing;
00024 using SixLabors.ImageSharp;
00025 using System.Reflection;
00026
00027 namespace VectSharp.Raster.ImageSharp
00028 {
00029     // Adapted from
      https://github.com/SixLabors/ImageSharp.Drawing/blob/master/src/ImageSharp.Drawing/Processing/GradientBrush.cs
      - Why isn't this public?  D:
00030     internal abstract class GradientBrushApplicator<TPixel> :  BrushApplicator<TPixel>
00031               where TPixel :  unmanaged, IPixel<TPixel>
00032     {
00033         private static readonly TPixel Transparent = Color.Transparent.ToPixel<TPixel>();
00034
00035         private readonly ColorStop[] colorStops;
00036
00037         private readonly GradientRepetitionMode repetitionMode;
00038
00039         private readonly MemoryAllocator allocator;
00040
00041         private readonly int scalineWidth;
00042
00043         private readonly object blenderBuffers;
00044
00045         private bool isDisposed;
00046
00047         private PixelBlender<TPixel> _blender;
00048
00049         IMemoryOwner<float> amountBuffer;
00050         IMemoryOwner<TPixel> overlayBuffer;
00051         static Type constructedLocalBlenderBuffersType;
00052
00053         static GradientBrushApplicator()
00054         {
00055             Assembly utilities = Assembly.Load("SixLabors.ImageSharp.Drawing");
00056             Type genericThreadLocalBlenderBuffersType =
      utilities.GetType("SixLabors.ImageSharp.Drawing.Utilities.ThreadLocalBlenderBuffers`1", true);
00057
00058             Type[] typeArgs = new Type[] { typeof(TPixel) };
00059             constructedLocalBlenderBuffersType =
      genericThreadLocalBlenderBuffersType.MakeGenericType(typeArgs);
```

```
00060         }
00061
00062
00063 /// <summary>
00064 /// Initializes a new instance of the <see cref="GradientBrushApplicator{TPixel}"/> class.
00065 /// </summary>
00066 /// <param name="configuration">The configuration instance to use when performing operations.</param>
00067 /// <param name="options">The graphics options.</param>
00068 /// <param name="target">The target image.</param>
00069 /// <param name="colorStops">An array of color stops sorted by their position.</param>
00070 /// <param name="repetitionMode">Defines if and how the gradient should be repeated.</param>
00071         protected GradientBrushApplicator(
00072             Configuration configuration,
00073             GraphicsOptions options,
00074             ImageFrame<TPixel> target,
00075             ColorStop[] colorStops,
00076             GradientRepetitionMode repetitionMode)
00077             :  base(configuration, options, target)
00078         {
00079             // TODO: requires colorStops to be sorted by position.
00080             // Use Array.Sort with a custom comparer.
00081             this.colorStops = colorStops;
00082             this.repetitionMode = repetitionMode;
00083             this.scalineWidth = target.Width;
00084             this.allocator = configuration.MemoryAllocator;
00085
00086             this.blenderBuffers = Activator.CreateInstance(constructedLocalBlenderBuffersType, new
      object[] { this.allocator, this.scalineWidth, false });
00087
00088             this._blender = (PixelBlender<TPixel>)this.GetType().GetProperty("Blender",
      BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.GetProperty).GetValue(this);
00089
00090             object data = this.blenderBuffers.GetType().GetField("data", BindingFlags.Instance |
      BindingFlags.NonPublic | BindingFlags.GetField).GetValue(this.blenderBuffers);
00091             object bufferOwner = data.GetType().GetProperty("Value", BindingFlags.Instance |
      BindingFlags.Public | BindingFlags.GetProperty).GetValue(data);
00092             amountBuffer = (IMemoryOwner<float>)bufferOwner.GetType().GetField("amountBuffer",
      BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.GetField).GetValue(bufferOwner);
00093             overlayBuffer = (IMemoryOwner<TPixel>)bufferOwner.GetType().GetField("overlayBuffer",
      BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.GetField).GetValue(bufferOwner);
00094         }
00095
00096         internal TPixel this[int x, int y]
00097         {
00098             get
00099             {
00100                 float positionOnCompleteGradient = this.PositionOnGradient(x + 0.5f, y + 0.5f);
00101
00102                 switch (this.repetitionMode)
00103                 {
00104                     case GradientRepetitionMode.None:
00105                         // do nothing.  The following could be done, but is not necessary:
00106                         // onLocalGradient = Math.Min(0, Math.Max(1, onLocalGradient));
00107                         break;
00108                     case GradientRepetitionMode.Repeat:
00109                         positionOnCompleteGradient %= 1;
00110                         break;
00111                     case GradientRepetitionMode.Reflect:
00112                         positionOnCompleteGradient %= 2;
00113                         if (positionOnCompleteGradient > 1)
00114                         {
00115                             positionOnCompleteGradient = 2 - positionOnCompleteGradient;
00116                         }
00117
00118                         break;
00119                     case GradientRepetitionMode.DontFill:
00120                         if (positionOnCompleteGradient > 1 || positionOnCompleteGradient < 0)
00121                         {
00122                             return Transparent;
00123                         }
00124
00125                         break;
00126                     default:
00127                         throw new ArgumentOutOfRangeException();
00128                 }
00129
00130                 (ColorStop from, ColorStop to) = this.GetGradientSegment(positionOnCompleteGradient);
00131
00132                 if (from.Color.Equals(to.Color))
00133                 {
00134                     return from.Color.ToPixel<TPixel>();
00135                 }
00136
00137                 float onLocalGradient = (positionOnCompleteGradient - from.Ratio) / (to.Ratio -
      from.Ratio);
00138
00139                 return new Color(Vector4.Lerp((Vector4)from.Color, (Vector4)to.Color,
```

```
onLocalGradient)).ToPixel<TPixel>();
00140             }
00141         }
00142 
00143 /// <inheritdoc />
00144         public override void Apply(Span<float> scanline, int x, int y)
00145         {
00146             Span<float> amounts = amountBuffer.Memory.Span.Slice(0, scanline.Length);
00147             Span<TPixel> overlays = overlayBuffer.Memory.Span.Slice(0, scanline.Length);
00148 
00149             float blendPercentage = this.Options.BlendPercentage;
00150 
00151             // TODO: Remove bounds checks.
00152             if (blendPercentage < 1)
00153             {
00154                 for (int i = 0; i < scanline.Length; i++)
00155                 {
00156                     amounts[i] = scanline[i] * blendPercentage;
00157                     overlays[i] = this[x + i, y];
00158                 }
00159             }
00160             else
00161             {
00162                 for (int i = 0; i < scanline.Length; i++)
00163                 {
00164                     amounts[i] = scanline[i];
00165                     overlays[i] = this[x + i, y];
00166                 }
00167             }
00168 
00169             Span<TPixel> destinationRow = this.Target.PixelBuffer.DangerousGetRowSpan(y).Slice(x,
scanline.Length);
00170 
00171             this._blender.Blend(this.Configuration, destinationRow, destinationRow, overlays,
amounts);
00172         }
00173 
00174 /// <summary>
00175 /// Calculates the position on the gradient for a given point.
00176 /// This method is abstract as it's content depends on the shape of the gradient.
00177 /// </summary>
00178 /// <param name="x">The x-coordinate of the point.</param>
00179 /// <param name="y">The y-coordinate of the point.</param>
00180 /// <returns>
00181 /// The position the given point has on the gradient.
00182 /// The position is not bound to the [0..1] interval.
00183 /// Values outside of that interval may be treated differently,
00184 /// e.g.  for the <see cref="GradientRepetitionMode" /> enum.
00185 /// </returns>
00186         protected abstract float PositionOnGradient(float x, float y);
00187 
00188 /// <inheritdoc/>
00189         protected override void Dispose(bool disposing)
00190         {
00191             if (this.isDisposed)
00192             {
00193                 return;
00194             }
00195 
00196             base.Dispose(disposing);
00197 
00198             if (disposing)
00199             {
00200                 ((IDisposable)this.blenderBuffers).Dispose();
00201             }
00202 
00203             this.isDisposed = true;
00204         }
00205 
00206         private (ColorStop From, ColorStop To) GetGradientSegment(float positionOnCompleteGradient)
00207         {
00208             ColorStop localGradientFrom = this.colorStops[0];
00209             ColorStop localGradientTo = default;
00210 
00211             // TODO: ensure colorStops has at least 2 items (technically 1 would be okay, but that's
no gradient)
00212             foreach (ColorStop colorStop in this.colorStops)
00213             {
00214                 localGradientTo = colorStop;
00215 
00216                 if (colorStop.Ratio > positionOnCompleteGradient)
00217                 {
00218                     // we're done here, so break it!
00219                     break;
00220                 }
00221 
00222                 localGradientFrom = localGradientTo;
```

```
00223             }
00224
00225             return (localGradientFrom, localGradientTo);
00226         }
00227     }
00228 }
```

8.59 ImageSharpContext.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using SixLabors.ImageSharp;
00020 using SixLabors.ImageSharp.Drawing;
00021 using SixLabors.ImageSharp.Processing;
00022 using SixLabors.ImageSharp.Drawing.Processing;
00023 using System.Collections.Generic;
00024 using SixLabors.ImageSharp.Advanced;
00025 using System.IO;
00026 using VectSharp.Filters;
00027 using SixLabors.ImageSharp.Formats.Gif;
00028 using System.Threading.Tasks;
00029 using System.Linq;
00030
00031 namespace VectSharp.Raster.ImageSharp
00032 {
00033     internal static class MatrixUtils
00034     {
00035         public static System.Numerics.Matrix3x2 ToMatrix(this double[,] matrix)
00036         {
00037             return new System.Numerics.Matrix3x2((float)matrix[0, 0], (float)matrix[1, 0],
      (float)matrix[0, 1], (float)matrix[1, 1], (float)matrix[0, 2], (float)matrix[1, 2]);
00038         }
00039
00040         public static double[] Multiply(double[,] matrix, double[] vector)
00041         {
00042             double[] tbr = new double[2];
00043
00044             tbr[0] = matrix[0, 0] * vector[0] + matrix[0, 1] * vector[1] + matrix[0, 2];
00045             tbr[1] = matrix[1, 0] * vector[0] + matrix[1, 1] * vector[1] + matrix[1, 2];
00046
00047             return tbr;
00048         }
00049
00050         public static Point Multiply(this double[,] matrix, Point vector)
00051         {
00052             return new Point(matrix[0, 0] * vector.X + matrix[0, 1] * vector.Y + matrix[0, 2],
      matrix[1, 0] * vector.X + matrix[1, 1] * vector.Y + matrix[1, 2]);
00053         }
00054
00055         public static double[,] Multiply(double[,] matrix1, double[,] matrix2)
00056         {
00057             double[,] tbr = new double[3, 3];
00058
00059             for (int i = 0; i < 3; i++)
00060             {
00061                 for (int j = 0; j < 3; j++)
00062                 {
00063                     for (int k = 0; k < 3; k++)
00064                     {
00065                         tbr[i, j] += matrix1[i, k] * matrix2[k, j];
00066                     }
00067                 }
00068             }
00069
00070             return tbr;
00071         }
00072
```

```
00073         public static double[,] Rotate(double[,] matrix, double angle)
00074         {
00075             double[,] rotationMatrix = new double[3, 3];
00076             rotationMatrix[0, 0] = Math.Cos(angle);
00077             rotationMatrix[0, 1] = -Math.Sin(angle);
00078             rotationMatrix[1, 0] = Math.Sin(angle);
00079             rotationMatrix[1, 1] = Math.Cos(angle);
00080             rotationMatrix[2, 2] = 1;
00081 
00082             return Multiply(matrix, rotationMatrix);
00083         }
00084 
00085         public static double[,] Translate(double[,] matrix, double x, double y)
00086         {
00087             double[,] translationMatrix = new double[3, 3];
00088             translationMatrix[0, 0] = 1;
00089             translationMatrix[0, 2] = x;
00090             translationMatrix[1, 1] = 1;
00091             translationMatrix[1, 2] = y;
00092             translationMatrix[2, 2] = 1;
00093 
00094             return Multiply(matrix, translationMatrix);
00095         }
00096 
00097         public static double[,] Scale(double[,] matrix, double scaleX, double scaleY)
00098         {
00099             double[,] scaleMatrix = new double[3, 3];
00100             scaleMatrix[0, 0] = scaleX;
00101             scaleMatrix[1, 1] = scaleY;
00102             scaleMatrix[2, 2] = 1;
00103 
00104             return Multiply(matrix, scaleMatrix);
00105         }
00106 
00107         public static double[,] Identity = new double[,] { { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0, 1 } };
00108         public static double[,] Invert(double[,] m)
00109         {
00110             double[,] tbr = new double[3, 3];
00111 
00112             tbr[0, 0] = (m[1, 1] * m[2, 2] - m[1, 2] * m[2, 1]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00113             tbr[0, 1] = -(m[0, 1] * m[2, 2] - m[0, 2] * m[2, 1]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00114             tbr[0, 2] = (m[0, 1] * m[1, 2] - m[0, 2] * m[1, 1]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00115             tbr[1, 0] = -(m[1, 0] * m[2, 2] - m[1, 2] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00116             tbr[1, 1] = (m[0, 0] * m[2, 2] - m[0, 2] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00117             tbr[1, 2] = -(m[0, 0] * m[1, 2] - m[0, 2] * m[1, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00118             tbr[2, 0] = (m[1, 0] * m[2, 1] - m[1, 1] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00119             tbr[2, 1] = -(m[0, 0] * m[2, 1] - m[0, 1] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00120             tbr[2, 2] = (m[0, 0] * m[1, 1] - m[0, 1] * m[1, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00121 
00122             return tbr;
00123         }
00124 
00125         public static double Determinant(double[,] matrix)
00126         {
00127             return (matrix[0, 0] * matrix[1, 1] - matrix[1, 0] * matrix[0, 1]) * matrix[2, 2] -
      (matrix[0, 0] * matrix[1, 2] - matrix[1, 0] * matrix[0, 2]) * matrix[2, 1] + (matrix[0, 1] * matrix[1,
      2] - matrix[1, 1] * matrix[0, 2]) * matrix[2, 0];
00128         }
00129     }
00130 
00131     internal class ImageSharpContext :  IGraphicsContext
00132     {
00133         public Image<SixLabors.ImageSharp.PixelFormats.Rgba32> Image { get; }
00134         private Image Buffer { get; set; }
00135         private Image ClipBuffer { get; }
00136 
00137         public double Width { get; private set; }
00138 
00139         public double Height { get; private set; }
```

```
00140
00141         public int IntWidth { get; private set; }
00142         public int IntHeight { get; private set; }
00143
00144         public Font Font { get; set; }
00145         public TextBaselines TextBaseline { get; set; }
00146
00147         public Brush FillStyle { get; set; }
00148
00149         public Brush StrokeStyle { get; set; }
00150
00151         public double LineWidth { get; set; }
00152         public LineCaps LineCap { get; set; }
00153         public LineJoins LineJoin { get; set; }
00154
00155         public LineDash Dash { get; set; }
00156         public string Tag { get; set; }
00157
00158         private double[,] _transform;
00159         private readonly Stack<double[,]> states;
00160
00161         private readonly Stack<Image> clips;
00162         private Image CurrentClip;
00163
00164         private List<IDisposable> disposables;
00165
00166         PathBuilder currentPath;
00167         bool figureInitialised;
00168         Point currentPoint;
00169         double scaleFactor;
00170
00171         public ImageSharpContext(double width, double height, double scaleFactor, Colour
      backgroundColour)
00172         {
00173             currentPath = new PathBuilder();
00174             figureInitialised = false;
00175             currentPoint = new Point();
00176
00177             IntWidth = (int)(width * scaleFactor);
00178             IntHeight = (int)(height * scaleFactor);
00179
00180             this.scaleFactor = Math.Sqrt(IntWidth / width * IntHeight / height);
00181
00182             _transform = new double[3, 3];
00183
00184             _transform[0, 0] = scaleFactor;
00185             _transform[1, 1] = scaleFactor;
00186             _transform[2, 2] = 1;
00187
00188             states = new Stack<double[,]>();
00189
00190             Width = width;
00191             Height = height;
00192
00193             this.Image = new Image<SixLabors.ImageSharp.PixelFormats.Rgba32>(IntWidth, IntHeight);
00194             this.Image.Mutate(x => x.Fill(backgroundColour.ToImageSharpColor()));
00195
00196             this.Buffer = new Image<SixLabors.ImageSharp.PixelFormats.Rgba32>(IntWidth, IntHeight);
00197             this.Buffer.Mutate(x => x.Clear(Color.FromRgba(0, 0, 0, 0)));
00198
00199             this.ClipBuffer = new Image<SixLabors.ImageSharp.PixelFormats.Rgba32>(IntWidth,
      IntHeight);
00200             this.ClipBuffer.Mutate(x => x.Clear(Color.FromRgba(0, 0, 0, 0)));
00201
00202             this.clips = new Stack<Image>();
00203             this.CurrentClip = null;
00204
00205             disposables = new List<IDisposable>();
00206         }
00207
00208         public void DisposeAllExceptImage()
00209         {
00210             for (int i = 0; i < disposables.Count; i++)
00211             {
00212                 disposables[i].Dispose();
00213                 this.CurrentClip?.Dispose();
00214                 this.Buffer.Dispose();
00215                 this.ClipBuffer.Dispose();
00216
00217             }
00218         }
00219
00220         public void Close()
00221         {
00222             this.currentPath.CloseFigure();
00223             this.figureInitialised = false;
00224         }
```

```
00225
00226         public void CubicBezierTo(double p1X, double p1Y, double p2X, double p2Y, double p3X, double
      p3Y)
00227         {
00228             Utils.CoerceNaNAndInfinityToZero(ref p1X, ref p1Y, ref p2X, ref p2Y, ref p3X, ref p3Y);
00229
00230             if (figureInitialised)
00231             {
00232                 Point p1 = new Point(p1X, p1Y);
00233                 Point p2 = new Point(p2X, p2Y);
00234                 Point p3 = new Point(p3X, p3Y);
00235                 currentPath.AddCubicBezier(currentPoint.ToPointF(1), p1.ToPointF(1), p2.ToPointF(1),
      p3.ToPointF(1));
00236                 currentPoint = p3;
00237             }
00238             else
00239             {
00240                 currentPath.StartFigure();
00241                 currentPoint = new Point(p3X, p3Y);
00242                 figureInitialised = true;
00243             }
00244         }
00245
00246         public void DrawRasterImage(int sourceX, int sourceY, int sourceWidth, int sourceHeight,
      double destinationX, double destinationY, double destinationWidth, double destinationHeight,
      RasterImage image)
00247         {
00248             Image sourceImage;
00249
00250             unsafe
00251             {
00252                 if (image.HasAlpha)
00253                 {
00254                     ReadOnlySpan<byte> data = new ReadOnlySpan<byte>((void*)image.ImageDataAddress,
      image.Width * image.Height * 4);
00255                     sourceImage =
      SixLabors.ImageSharp.Image.LoadPixelData<SixLabors.ImageSharp.PixelFormats.Rgba32>(data, image.Width,
      image.Height);
00256                 }
00257                 else
00258                 {
00259                     ReadOnlySpan<byte> data = new ReadOnlySpan<byte>((void*)image.ImageDataAddress,
      image.Width * image.Height * 3);
00260                     sourceImage =
      SixLabors.ImageSharp.Image.LoadPixelData<SixLabors.ImageSharp.PixelFormats.Rgb24>(data, image.Width,
      image.Height);
00261                 }
00262             }
00263
00264             DrawImage(sourceX, sourceY, sourceWidth, sourceHeight, destinationX, destinationY,
      destinationWidth, destinationHeight, image.Interpolate, sourceImage);
00265         }
00266
00267         internal void DrawImage(int sourceX, int sourceY, int sourceWidth, int sourceHeight, double
      destinationX, double destinationY, double destinationWidth, double destinationHeight, bool
      interpolate, Image sourceImage)
00268         {
00269             sourceImage.Mutate(x => x.Crop(new SixLabors.ImageSharp.Rectangle(sourceX, sourceY,
      sourceWidth, sourceHeight)));
00270
00271             Point targetPoint = _transform.Multiply(new Point(destinationX, destinationY));
00272             Point targetPointX = _transform.Multiply(new Point(destinationX + destinationWidth,
      destinationY));
00273             Point targetPointY = _transform.Multiply(new Point(destinationX, destinationY +
      destinationHeight));
00274             Point targetPointXY = _transform.Multiply(new Point(destinationX + destinationWidth,
      destinationY + destinationHeight));
00275
00276             double minX = Math.Min(Math.Min(targetPoint.X, targetPointX.X), Math.Min(targetPointY.X,
      targetPointXY.X));
00277             double minY = Math.Min(Math.Min(targetPoint.Y, targetPointX.Y), Math.Min(targetPointY.Y,
      targetPointXY.Y));
00278
00279             double maxX = Math.Max(Math.Max(targetPoint.X, targetPointX.X), Math.Max(targetPointY.X,
      targetPointXY.X));
00280             double maxY = Math.Max(Math.Max(targetPoint.Y, targetPointX.Y), Math.Max(targetPointY.Y,
      targetPointXY.Y));
00281
00282             double[,] currTransform = MatrixUtils.Multiply(MatrixUtils.Translate(_transform,
      destinationX, destinationY), MatrixUtils.Scale(MatrixUtils.Identity, destinationWidth /
      sourceImage.Width, destinationHeight / sourceImage.Height));
00283
00284             Point origin = currTransform.Multiply(new Point(0, 0));
00285
00286             double[,] translation = MatrixUtils.Translate(MatrixUtils.Identity, -minX, -minY);
00287
00288             double[,] centeredTransform = MatrixUtils.Multiply(translation, currTransform);
```

```
00289
00290             SixLabors.ImageSharp.Processing.Processors.Transforms.IResampler resampler;
00291
00292             if (interpolate)
00293             {
00294                 resampler = KnownResamplers.Bicubic;
00295             }
00296             else
00297             {
00298                 resampler = KnownResamplers.NearestNeighbor;
00299             }
00300
00301             sourceImage.Mutate(x => x.Transform(new SixLabors.ImageSharp.Rectangle(sourceX, sourceY,
      sourceWidth, sourceHeight), centeredTransform.ToMatrix(), new
      SixLabors.ImageSharp.Size((int)Math.Round(maxX - minX), (int)Math.Round(maxY - minY)), resampler));
00302
00303             if (this.CurrentClip == null)
00304             {
00305                 this.Image.Mutate(x => x.CheckedDrawImage(sourceImage, new
      SixLabors.ImageSharp.Point((int)Math.Round(minX), (int)Math.Round(minY)), 1));
00306             }
00307             else
00308             {
00309                 this.ClipBuffer.Mutate(x => x.Clear(Color.FromRgba(0, 0, 0, 0)));
00310                 this.ClipBuffer.Mutate(x => x.CheckedDrawImage(sourceImage, new
      SixLabors.ImageSharp.Point((int)Math.Round(minX), (int)Math.Round(minY)), 1));
00311
00312                 GraphicsOptions opt = new GraphicsOptions() { AlphaCompositionMode =
      SixLabors.ImageSharp.PixelFormats.PixelAlphaCompositionMode.SrcIn };
00313
00314                 this.Buffer.Dispose();
00315
00316                 this.Buffer = this.CurrentClip.Clone(x => x.CheckedDrawImage(this.ClipBuffer, opt));
00317
00318                 this.Image.Mutate(x => x.CheckedDrawImage(this.Buffer, 1));
00319             }
00320         }
00321
00322         public void Fill(FillRule fillRule)
00323         {
00324             IPath path = this.currentPath.Build();
00325
00326             ShapeOptions shapeOptions = new ShapeOptions();
00327
00328             switch (fillRule)
00329             {
00330                 case FillRule.NonZeroWinding:
00331                     shapeOptions.IntersectionRule = IntersectionRule.NonZero;
00332                     break;
00333                 case FillRule.EvenOdd:
00334                     shapeOptions.IntersectionRule = IntersectionRule.EvenOdd;
00335                     break;
00336
00337             }
00338
00339             if (this.CurrentClip == null)
00340             {
00341                 this.Image.Mutate(x => x.Fill(new DrawingOptions() { Transform =
      _transform.ToMatrix(), ShapeOptions = shapeOptions }, this.FillStyle.ToImageSharpBrush(_transform),
      path));
00342             }
00343             else
00344             {
00345                 this.ClipBuffer.Mutate(x => x.Clear(Color.FromRgba(0, 0, 0, 0)));
00346                 this.ClipBuffer.Mutate(x => x.Fill(new DrawingOptions() { Transform =
      _transform.ToMatrix(), ShapeOptions = shapeOptions }, this.FillStyle.ToImageSharpBrush(_transform),
      path));
00347
00348                 GraphicsOptions opt = new GraphicsOptions() { AlphaCompositionMode =
      SixLabors.ImageSharp.PixelFormats.PixelAlphaCompositionMode.SrcIn };
00349
00350                 this.Buffer.Dispose();
00351
00352                 this.Buffer = this.CurrentClip.Clone(x => x.CheckedDrawImage(this.ClipBuffer, opt));
00353
00354                 this.Image.Mutate(x => x.CheckedDrawImage(this.Buffer, 1));
00355             }
00356
00357             this.currentPath = new PathBuilder();
00358             this.figureInitialised = false;
00359         }
00360
00361         public void FillText(string text, double x, double y)
00362         {
00363             PathText(text, x, y);
00364             Fill(FillRule.NonZeroWinding);
00365         }
```

```
00366
00367         private void PathText(string text, double x, double y)
00368         {
00369             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00370
00371             GraphicsPath textPath = new GraphicsPath().AddText(x, y, text, Font, TextBaseline);
00372
00373             for (int j = 0; j < textPath.Segments.Count; j++)
00374             {
00375                 switch (textPath.Segments[j].Type)
00376                 {
00377                     case VectSharp.SegmentType.Move:
00378                         this.MoveTo(textPath.Segments[j].Point.X, textPath.Segments[j].Point.Y);
00379                         break;
00380                     case VectSharp.SegmentType.Line:
00381                         this.LineTo(textPath.Segments[j].Point.X, textPath.Segments[j].Point.Y);
00382                         break;
00383                     case VectSharp.SegmentType.CubicBezier:
00384                         this.CubicBezierTo(textPath.Segments[j].Points[0].X,
      textPath.Segments[j].Points[0].Y, textPath.Segments[j].Points[1].X, textPath.Segments[j].Points[1].Y,
      textPath.Segments[j].Points[2].X, textPath.Segments[j].Points[2].Y);
00385                         break;
00386                     case VectSharp.SegmentType.Close:
00387                         this.Close();
00388                         break;
00389                 }
00390             }
00391         }
00392
00393         public void LineTo(double x, double y)
00394         {
00395             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00396
00397             if (figureInitialised)
00398             {
00399                 Point newPoint = new Point(x, y);
00400                 currentPath.AddLine(currentPoint.ToPointF(1), newPoint.ToPointF(1));
00401                 currentPoint = newPoint;
00402             }
00403             else
00404             {
00405                 currentPath.StartFigure();
00406                 currentPoint = new Point(x, y);
00407                 figureInitialised = true;
00408             }
00409         }
00410
00411         public void MoveTo(double x, double y)
00412         {
00413             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00414
00415             currentPath.StartFigure();
00416             currentPoint = new Point(x, y);
00417             figureInitialised = true;
00418         }
00419
00420         public void Rectangle(double x0, double y0, double width, double height)
00421         {
00422             MoveTo(x0, y0);
00423             LineTo(x0 + width, y0);
00424             LineTo(x0 + width, y0 + height);
00425             LineTo(x0, y0 + height);
00426             Close();
00427         }
00428
00429         public void Restore()
00430         {
00431             _transform = states.Pop();
00432
00433             Image newClip = clips.Pop();
00434             if (CurrentClip != newClip)
00435             {
00436                 CurrentClip.Dispose();
00437             }
00438
00439             CurrentClip = newClip;
00440         }
00441
00442         public void Rotate(double angle)
00443         {
00444             Utils.CoerceNaNAndInfinityToZero(ref angle);
00445
00446             _transform = MatrixUtils.Rotate(_transform, angle);
00447
00448             currentPath = new PathBuilder();
00449             figureInitialised = false;
00450         }
```

```
00451
00452         public void Save()
00453         {
00454             states.Push((double[,])_transform.Clone());
00455             clips.Push(CurrentClip);
00456         }
00457
00458         public void Scale(double x, double y)
00459         {
00460             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00461
00462             _transform = MatrixUtils.Scale(_transform, x, y);
00463
00464             currentPath = new PathBuilder();
00465             figureInitialised = false;
00466         }
00467
00468         public void SetClippingPath()
00469         {
00470             IPath path = this.currentPath.Build();
00471
00472             Image newClip;
00473
00474             DrawingOptions opt = new DrawingOptions() { Transform = _transform.ToMatrix() };
00475
00476             if (this.CurrentClip == null)
00477             {
00478                 newClip = new Image<SixLabors.ImageSharp.PixelFormats.Rgba32>(IntWidth, IntHeight);
00479                 newClip.Mutate(x => x.Clear(Color.FromRgba(0, 0, 0, 0)));
00480                 newClip.Mutate(x => x.Fill(opt, Color.FromRgb(0, 0, 0), path));
00481             }
00482             else
00483             {
00484                 this.Buffer.Mutate(x => x.Clear(Color.FromRgba(0, 0, 0, 0)));
00485                 this.Buffer.Mutate(x => x.Fill(opt, Color.FromRgb(0, 0, 0), path));
00486
00487                 newClip = this.CurrentClip.Clone(x => x.CheckedDrawImage(this.Buffer, new
      GraphicsOptions() { AlphaCompositionMode =
      SixLabors.ImageSharp.PixelFormats.PixelAlphaCompositionMode.SrcIn }));
00488
00489                 disposables.Add(this.CurrentClip);
00490             }
00491
00492             this.CurrentClip = newClip;
00493
00494             this.currentPath = new PathBuilder();
00495             this.figureInitialised = false;
00496         }
00497
00498         public void SetFillStyle((int r, int g, int b, double a) style)
00499         {
00500             this.FillStyle = Colour.FromRgba(style);
00501         }
00502
00503         public void SetFillStyle(Brush style)
00504         {
00505             this.FillStyle = style;
00506         }
00507
00508         public void SetLineDash(LineDash dash)
00509         {
00510             this.Dash = dash;
00511         }
00512
00513         public void SetStrokeStyle((int r, int g, int b, double a) style)
00514         {
00515             this.StrokeStyle = Colour.FromRgba(style);
00516         }
00517
00518         public void SetStrokeStyle(Brush style)
00519         {
00520             this.StrokeStyle = style;
00521         }
00522
00523         public void Stroke()
00524         {
00525             if (this.LineWidth > 0)
00526             {
00527                 if (this.CurrentClip == null)
00528                 {
00529                     this.Image.Mutate(x => x.Fill(new DrawingOptions() { Transform =
      _transform.ToMatrix() }, this.StrokeStyle.ToImageSharpBrush(_transform),
      this.currentPath.Build().GenerateOutline((float)(this.LineWidth),
      this.Dash.ToImageSharpDash(this.LineWidth), false, this.LineJoin.ToImageSharpJoint(),
      this.LineCap.ToImageSharpCap())));
00530                 }
00531                 else
```

```
00532                 {
00533                     this.ClipBuffer.Mutate(x => x.Clear(Color.FromRgba(0, 0, 0, 0)));
00534                     this.ClipBuffer.Mutate(x => x.Fill(new DrawingOptions() { Transform =
      _transform.ToMatrix() }, this.StrokeStyle.ToImageSharpBrush(_transform),
      this.currentPath.Build().GenerateOutline((float)(this.LineWidth),
      this.Dash.ToImageSharpDash(this.LineWidth), false, this.LineJoin.ToImageSharpJoint(),
      this.LineCap.ToImageSharpCap())));
00535
00536                     GraphicsOptions opt = new GraphicsOptions() { AlphaCompositionMode =
      SixLabors.ImageSharp.PixelFormats.PixelAlphaCompositionMode.SrcIn };
00537
00538                     this.Buffer.Dispose();
00539
00540                     this.Buffer = this.CurrentClip.Clone(x => x.CheckedDrawImage(this.ClipBuffer,
      opt));
00541
00542                     this.Image.Mutate(x => x.CheckedDrawImage(this.Buffer, 1));
00543                 }
00544             }
00545
00546             this.currentPath = new PathBuilder();
00547             this.figureInitialised = false;
00548         }
00549
00550         public void StrokeText(string text, double x, double y)
00551         {
00552             PathText(text, x, y);
00553             Stroke();
00554         }
00555
00556         public void Transform(double a, double b, double c, double d, double e, double f)
00557         {
00558             Utils.CoerceNaNAndInfinityToZero(ref a, ref b, ref c, ref d, ref e, ref f);
00559
00560             double[,] transfMatrix = new double[3, 3] { { a, c, e }, { b, d, f }, { 0, 0, 1 } };
00561             _transform = MatrixUtils.Multiply(_transform, transfMatrix);
00562
00563             currentPath = new PathBuilder();
00564             figureInitialised = false;
00565         }
00566
00567         public void Translate(double x, double y)
00568         {
00569             Utils.CoerceNaNAndInfinityToZero(ref x, ref y);
00570
00571             _transform = MatrixUtils.Translate(_transform, x, y);
00572
00573             currentPath = new PathBuilder();
00574             figureInitialised = false;
00575         }
00576
00577         public void DrawFilteredGraphics(Graphics graphics, IFilter filter)
00578         {
00579             double scale = this.scaleFactor * Math.Sqrt(MatrixUtils.Determinant(_transform));
00580
00581             Rectangle bounds = graphics.GetBounds();
00582
00583             bounds = new Rectangle(bounds.Location.X - filter.TopLeftMargin.X, bounds.Location.Y -
      filter.TopLeftMargin.Y, bounds.Size.Width + filter.TopLeftMargin.X + filter.BottomRightMargin.X,
      bounds.Size.Height + filter.TopLeftMargin.Y + filter.BottomRightMargin.Y);
00584
00585             if (bounds.Size.Width > 0 && bounds.Size.Height > 0)
00586             {
00587                 bool rasterisationNeeded = true;
00588
00589                 if (filter is GaussianBlurFilter gauss)
00590                 {
00591                     rasterisationNeeded = false;
00592
00593                     Page pag = new Page(1, 1);
00594                     pag.Graphics.DrawGraphics(0, 0, graphics);
00595                     pag.Crop(bounds.Location, bounds.Size);
00596
00597                     Image<SixLabors.ImageSharp.PixelFormats.Rgba32> img =
      ImageSharpContextInterpreter.SaveAsImage(pag, scale);
00598                     img.Mutate(x => x.GaussianBlur((float)(gauss.StandardDeviation * scale)));
00599
00600                     DrawImage(0, 0, img.Width, img.Height, bounds.Location.X, bounds.Location.Y,
      bounds.Size.Width, bounds.Size.Height, true, img);
00601
00602                     img.Dispose();
00603                 }
00604                 else if (filter is ColourMatrixFilter cmf)
00605                 {
00606                     rasterisationNeeded = false;
00607
00608                     Page pag = new Page(1, 1);
```

```
00609                     pag.Graphics.DrawGraphics(0, 0, graphics);
00610                     pag.Crop(bounds.Location, bounds.Size);
00611 
00612                     Image<SixLabors.ImageSharp.PixelFormats.Rgba32> img =
      ImageSharpContextInterpreter.SaveAsImage(pag, scale);
00613 
00614                     img.Mutate(x => x.Filter(new ColorMatrix((float)cmf.ColourMatrix.R1,
      (float)cmf.ColourMatrix.G1, (float)cmf.ColourMatrix.B1, (float)cmf.ColourMatrix.A1,
      (float)cmf.ColourMatrix.R2, (float)cmf.ColourMatrix.G2, (float)cmf.ColourMatrix.B2,
      (float)cmf.ColourMatrix.A2, (float)cmf.ColourMatrix.R3, (float)cmf.ColourMatrix.G3,
      (float)cmf.ColourMatrix.B3, (float)cmf.ColourMatrix.A3, (float)cmf.ColourMatrix.R4,
      (float)cmf.ColourMatrix.G4, (float)cmf.ColourMatrix.B4, (float)cmf.ColourMatrix.A4,
      (float)cmf.ColourMatrix.R5, (float)cmf.ColourMatrix.G5, (float)cmf.ColourMatrix.B5,
      (float)cmf.ColourMatrix.A5)));
00615 
00616                     DrawImage(0, 0, img.Width, img.Height, bounds.Location.X, bounds.Location.Y,
      bounds.Size.Width, bounds.Size.Height, true, img);
00617 
00618                     img.Dispose();
00619                 }
00620                 else if (filter is BoxBlurFilter bbf)
00621                 {
00622                     rasterisationNeeded = false;
00623 
00624                     Page pag = new Page(1, 1);
00625                     pag.Graphics.DrawGraphics(0, 0, graphics);
00626                     pag.Crop(bounds.Location, bounds.Size);
00627 
00628                     Image<SixLabors.ImageSharp.PixelFormats.Rgba32> img =
      ImageSharpContextInterpreter.SaveAsImage(pag, scale);
00629 
00630                     img.Mutate(x => x.BoxBlur((int)Math.Round(bbf.BoxRadius * scale)));
00631 
00632                     DrawImage(0, 0, img.Width, img.Height, bounds.Location.X, bounds.Location.Y,
      bounds.Size.Width, bounds.Size.Height, true, img);
00633 
00634                     img.Dispose();
00635                 }
00636                 else if (filter is CompositeLocationInvariantFilter comp)
00637                 {
00638                     bool allSupported = true;
00639 
00640                     foreach (IFilter filter2 in comp.Filters)
00641                     {
00642                         if (!(filter2 is GaussianBlurFilter) && !(filter2 is ColourMatrixFilter) &&
      !(filter2 is BoxBlurFilter))
00643                         {
00644                             allSupported = false;
00645                             break;
00646                         }
00647                     }
00648 
00649                     if (allSupported)
00650                     {
00651                         rasterisationNeeded = false;
00652 
00653                         Page pag = new Page(1, 1);
00654                         pag.Graphics.DrawGraphics(0, 0, graphics);
00655                         pag.Crop(bounds.Location, bounds.Size);
00656 
00657                         Image<SixLabors.ImageSharp.PixelFormats.Rgba32> img =
      ImageSharpContextInterpreter.SaveAsImage(pag, scale);
00658 
00659                         foreach (IFilter filter2 in comp.Filters)
00660                         {
00661                             if (filter2 is GaussianBlurFilter gauss2)
00662                             {
00663                                 img.Mutate(x => x.GaussianBlur((float)(gauss2.StandardDeviation *
      scale)));
00664                             }
00665                             else if (filter2 is ColourMatrixFilter cmf2)
00666                             {
00667                                 img.Mutate(x => x.Filter(new ColorMatrix((float)cmf2.ColourMatrix.R1,
      (float)cmf2.ColourMatrix.G1, (float)cmf2.ColourMatrix.B1, (float)cmf2.ColourMatrix.A1,
      (float)cmf2.ColourMatrix.R2, (float)cmf2.ColourMatrix.G2, (float)cmf2.ColourMatrix.B2,
      (float)cmf2.ColourMatrix.A2, (float)cmf2.ColourMatrix.R3, (float)cmf2.ColourMatrix.G3,
      (float)cmf2.ColourMatrix.B3, (float)cmf2.ColourMatrix.A3, (float)cmf2.ColourMatrix.R4,
      (float)cmf2.ColourMatrix.G4, (float)cmf2.ColourMatrix.B4, (float)cmf2.ColourMatrix.A4,
      (float)cmf2.ColourMatrix.R5, (float)cmf2.ColourMatrix.G5, (float)cmf2.ColourMatrix.B5,
      (float)cmf2.ColourMatrix.A5)));
00668                             }
00669                             else if (filter2 is BoxBlurFilter bbf2)
00670                             {
00671                                 img.Mutate(x => x.BoxBlur((int)Math.Round(bbf2.BoxRadius * scale)));
00672                             }
00673                         }
00674 
```

```
00675
00676                           DrawImage(0, 0, img.Width, img.Height, bounds.Location.X, bounds.Location.Y,
      bounds.Size.Width, bounds.Size.Height, true, img);
00677
00678                           img.Dispose();
00679                       }
00680                   }
00681
00682                   if (rasterisationNeeded)
00683                   {
00684                       RasterImage rasterised = ImageSharpContextInterpreter.Rasterise(graphics, bounds,
      scale, true);
00685                       RasterImage filtered = null;
00686
00687                       if (filter is IFilterWithRasterisableParameter filterWithRastParam)
00688                       {
00689                           filterWithRastParam.RasteriseParameter(ImageSharpContextInterpreter.Rasterise,
      scale);
00690                       }
00691
00692                       if (filter is ILocationInvariantFilter locInvFilter)
00693                       {
00694                           filtered = locInvFilter.Filter(rasterised, scale);
00695                       }
00696                       else if (filter is IFilterWithLocation filterWithLoc)
00697                       {
00698                           filtered = filterWithLoc.Filter(rasterised, bounds, scale);
00699                       }
00700
00701                       if (filtered != null)
00702                       {
00703                           rasterised.Dispose();
00704
00705                           DrawRasterImage(0, 0, filtered.Width, filtered.Height, bounds.Location.X,
      bounds.Location.Y, bounds.Size.Width, bounds.Size.Height, filtered);
00706                       }
00707                   }
00708               }
00709           }
00710       }
00711
00712       internal static class Utils
00713       {
00714           public static IImageProcessingContext CheckedDrawImage(this IImageProcessingContext x, Image
      image, SixLabors.ImageSharp.Point point, float opacity)
00715           {
00716               int x0A = 0;
00717               int y0A = 0;
00718               SixLabors.ImageSharp.Size size = x.GetCurrentSize();
00719               int x1A = size.Width;
00720               int y1A = size.Height;
00721
00722               int x0B = point.X;
00723               int y0B = point.Y;
00724               int x1B = point.X + image.Width;
00725               int y1B = point.Y + image.Height;
00726
00727               if (!(x0A >= x1B || x1A <= x0B || y0A >= y1B || y1A <= y0B))
00728               {
00729                   return x.DrawImage(image, point, opacity);
00730               }
00731               else
00732               {
00733                   return x;
00734               }
00735           }
00736
00737           public static IImageProcessingContext CheckedDrawImage(this IImageProcessingContext x, Image
      image, float opacity)
00738           {
00739               return x.DrawImage(image, opacity);
00740           }
00741
00742           public static IImageProcessingContext CheckedDrawImage(this IImageProcessingContext x, Image
      image, GraphicsOptions options)
00743           {
00744               return x.DrawImage(image, options);
00745           }
00746
00747           public static PointF ToPointF(this Point pt, double scaleFactor)
00748           {
00749               return new PointF((float)(pt.X * scaleFactor), (float)(pt.Y * scaleFactor));
00750           }
00751
00752           public static void CoerceNaNAndInfinityToZero(ref double val)
00753           {
00754               if (double.IsNaN(val) || double.IsInfinity(val) || val == double.MinValue || val ==
```

```
      double.MaxValue)
00755             {
00756                 val = 0;
00757             }
00758         }
00759
00760         public static void CoerceNaNAndInfinityToZero(ref double val1, ref double val2)
00761         {
00762             if (double.IsNaN(val1) || double.IsInfinity(val1) || val1 == double.MinValue || val1 ==
      double.MaxValue)
00763             {
00764                 val1 = 0;
00765             }
00766
00767             if (double.IsNaN(val2) || double.IsInfinity(val2) || val2 == double.MinValue || val2 ==
      double.MaxValue)
00768             {
00769                 val2 = 0;
00770             }
00771         }
00772
00773         public static void CoerceNaNAndInfinityToZero(ref double val1, ref double val2, ref double
      val3, ref double val4)
00774         {
00775             if (double.IsNaN(val1) || double.IsInfinity(val1) || val1 == double.MinValue || val1 ==
      double.MaxValue)
00776             {
00777                 val1 = 0;
00778             }
00779
00780             if (double.IsNaN(val2) || double.IsInfinity(val2) || val2 == double.MinValue || val2 ==
      double.MaxValue)
00781             {
00782                 val2 = 0;
00783             }
00784
00785             if (double.IsNaN(val3) || double.IsInfinity(val3) || val3 == double.MinValue || val3 ==
      double.MaxValue)
00786             {
00787                 val3 = 0;
00788             }
00789
00790             if (double.IsNaN(val4) || double.IsInfinity(val4) || val4 == double.MinValue || val4 ==
      double.MaxValue)
00791             {
00792                 val4 = 0;
00793             }
00794         }
00795
00796         public static void CoerceNaNAndInfinityToZero(ref double val1, ref double val2, ref double
      val3, ref double val4, ref double val5, ref double val6)
00797         {
00798             if (double.IsNaN(val1) || double.IsInfinity(val1) || val1 == double.MinValue || val1 ==
      double.MaxValue)
00799             {
00800                 val1 = 0;
00801             }
00802
00803             if (double.IsNaN(val2) || double.IsInfinity(val2) || val2 == double.MinValue || val2 ==
      double.MaxValue)
00804             {
00805                 val2 = 0;
00806             }
00807
00808             if (double.IsNaN(val3) || double.IsInfinity(val3) || val3 == double.MinValue || val3 ==
      double.MaxValue)
00809             {
00810                 val3 = 0;
00811             }
00812
00813             if (double.IsNaN(val4) || double.IsInfinity(val4) || val4 == double.MinValue || val4 ==
      double.MaxValue)
00814             {
00815                 val4 = 0;
00816             }
00817
00818             if (double.IsNaN(val5) || double.IsInfinity(val5) || val5 == double.MinValue || val5 ==
      double.MaxValue)
00819             {
00820                 val5 = 0;
00821             }
00822
00823             if (double.IsNaN(val6) || double.IsInfinity(val6) || val6 == double.MinValue || val6 ==
      double.MaxValue)
00824             {
00825                 val6 = 0;
00826             }
```

```
00827         }
00828
00829         public static Color ToImageSharpColor(this Colour colour)
00830         {
00831             return Color.FromRgba((byte)(colour.R * 255), (byte)(colour.G * 255), (byte)(colour.B *
      255), (byte)(colour.A * 255));
00832         }
00833
00834         public static SixLabors.ImageSharp.Drawing.Processing.Brush ToImageSharpBrush(this Brush
      brush, double[,] transform)
00835         {
00836             if (brush is SolidColourBrush solid)
00837             {
00838                 return new
      SixLabors.ImageSharp.Drawing.Processing.SolidBrush(solid.Colour.ToImageSharpColor());
00839             }
00840             else if (brush is LinearGradientBrush linear)
00841             {
00842                 ColorStop[] colorStops = new ColorStop[linear.GradientStops.Count];
00843
00844                 for (int i = 0; i < linear.GradientStops.Count; i++)
00845                 {
00846                     colorStops[i] = new ColorStop((float)linear.GradientStops[i].Offset,
      linear.GradientStops[i].Colour.ToImageSharpColor());
00847                 }
00848
00849                 return new
      SixLabors.ImageSharp.Drawing.Processing.LinearGradientBrush(transform.Multiply(linear.StartPoint).ToPointF(1),
      transform.Multiply(linear.EndPoint).ToPointF(1), GradientRepetitionMode.None, colorStops);
00850             }
00851             else if (brush is RadialGradientBrush radial)
00852             {
00853                 ColorStop[] colorStops = new ColorStop[radial.GradientStops.Count];
00854
00855                 for (int i = 0; i < radial.GradientStops.Count; i++)
00856                 {
00857                     colorStops[i] = new ColorStop((float)radial.GradientStops[i].Offset,
      radial.GradientStops[i].Colour.ToImageSharpColor());
00858                 }
00859
00860                 return new RadialGradientBrushSVGStyle(radial.Centre, radial.FocalPoint,
      radial.Radius, transform, GradientRepetitionMode.None, colorStops);
00861             }
00862             else
00863             {
00864                 throw new NotImplementedException();
00865             }
00866         }
00867
00868         public static float[] ToImageSharpDash(this LineDash dash, double lineThickness)
00869         {
00870             if (dash.DashArray != null && dash.DashArray.Any(x => x != 0))
00871             {
00872                 return dash.DashArray.Select(x => (float)(x /  lineThickness)).ToArray();
00873             }
00874             else
00875             {
00876                 return new float[] { };
00877             }
00878         }
00879
00880         public static JointStyle ToImageSharpJoint(this LineJoins join)
00881         {
00882             switch (join)
00883             {
00884                 case LineJoins.Round:
00885                     return JointStyle.Round;
00886                 case LineJoins.Miter:
00887                     return JointStyle.Miter;
00888                 case LineJoins.Bevel:
00889                     return JointStyle.Square;
00890                 default:
00891                     return JointStyle.Miter;
00892             }
00893         }
00894
00895         public static EndCapStyle ToImageSharpCap(this LineCaps cap)
00896         {
00897             switch (cap)
00898             {
00899                 case LineCaps.Square:
00900                     return EndCapStyle.Square;
00901                 case LineCaps.Round:
00902                     return EndCapStyle.Round;
00903                 case LineCaps.Butt:
00904                     return EndCapStyle.Butt;
00905                 default:
```

```
00906                     throw new NotImplementedException();
00907             }
00908         }
00909
00910         internal class RadialGradientBrushSVGStyle :
      SixLabors.ImageSharp.Drawing.Processing.GradientBrush
00911         {
00912             private readonly Point Centre;
00913             private readonly Point FocalPoint;
00914             private readonly double Radius;
00915             private readonly double[,] InverseTransform;
00916
00917             public RadialGradientBrushSVGStyle(Point centre, Point focalPoint, double radius,
      double[,] transform, GradientRepetitionMode repetitionMode, params ColorStop[] colorStops) :
      base(repetitionMode, colorStops)
00918             {
00919                 Centre = centre;
00920                 FocalPoint = focalPoint;
00921                 Radius = radius;
00922                 InverseTransform = MatrixUtils.Invert(transform);
00923             }
00924
00925             public override BrushApplicator<TPixel> CreateApplicator<TPixel>(Configuration
      configuration, GraphicsOptions options, ImageFrame<TPixel> source, RectangleF region)
00926             {
00927                 return new RadialGradientBrushPDFStyleApplicator<TPixel>(configuration, options,
      source, Centre, FocalPoint, Radius, InverseTransform, ColorStops, RepetitionMode);
00928             }
00929
00930             private class RadialGradientBrushPDFStyleApplicator<TPixel> :
      GradientBrushApplicator<TPixel> where TPixel :  unmanaged,
      SixLabors.ImageSharp.PixelFormats.IPixel<TPixel>
00931             {
00932                 private readonly Point Centre;
00933                 private readonly Point FocalPoint;
00934                 private readonly double Radius;
00935                 private readonly double[,] InverseTransform;
00936
00937                 private readonly double a;
00938
00939                 public RadialGradientBrushPDFStyleApplicator(
00940                     Configuration configuration,
00941                     GraphicsOptions options,
00942                     ImageFrame<TPixel> target,
00943                     Point centre, Point focalPoint, double radius, double[,] inverseTransform,
00944                     ColorStop[] colorStops,
00945                     GradientRepetitionMode repetitionMode)
00946                 :  base(configuration, options, target, colorStops, repetitionMode)
00947                 {
00948                     this.Centre = centre;
00949                     this.FocalPoint = focalPoint;
00950                     this.Radius = radius;
00951                     this.InverseTransform = inverseTransform;
00952
00953                     a = (centre.X - focalPoint.X) * (centre.X - focalPoint.X) + (centre.Y -
      focalPoint.Y) * (centre.Y - focalPoint.Y) - radius * radius;
00954                 }
00955
00956                 Random rnd = new Random();
00957
00958                 protected override float PositionOnGradient(float x, float y)
00959                 {
00960                     Point realPoint = InverseTransform.Multiply(new Point(x, y));
00961
00962                     double c = (realPoint.X - FocalPoint.X) * (realPoint.X - FocalPoint.X) +
      (realPoint.Y - FocalPoint.Y) * (realPoint.Y - FocalPoint.Y);
00963
00964                     double halfB = -((realPoint.X - FocalPoint.X) * (Centre.X - FocalPoint.X) +
      (realPoint.Y - FocalPoint.Y) * (Centre.Y - FocalPoint.Y));
00965
00966                     double sqrt = Math.Sqrt(halfB * halfB - a * c);
00967
00968                     double tbr1 = (-halfB + sqrt) / a;
00969                     double tbr2 = (-halfB - sqrt) / a;
00970
00971                     if (tbr1 >= 0 && tbr2 < 0)
00972                     {
00973                         return (float)tbr1;
00974                     }
00975                     else if (tbr1 < 0 && tbr2 >= 0)
00976                     {
00977                         return (float)tbr2;
00978                     }
00979                     else if (tbr1 < 0 && tbr2 < 0)
00980                     {
00981                         return 0;
00982                     }
```

```
00983                     else
00984                     {
00985                         return (float)Math.Min(tbr1, tbr2);
00986                     }
00987                 }
00988 
00989 /// <inheritdoc/>
00990                 public override void Apply(Span<float> scanline, int x, int y)
00991                 {
00992                     base.Apply(scanline, x, y);
00993                 }
00994             }
00995         }
00996     }
00997 
00998 /// <summary>
00999 /// Enumeration containing the supported output formats.
01000 /// </summary>
01001     public enum OutputFormats
01002     {
01003 /// <summary>
01004 /// Windows bitmap format
01005 /// </summary>
01006         BMP,
01007 
01008 /// <summary>
01009 /// Graphics interchange format
01010 /// </summary>
01011         GIF,
01012 
01013 /// <summary>
01014 /// Joint photographic experts group format
01015 /// </summary>
01016         JPEG,
01017 
01018 
01019 /// <summary>
01020 /// Portable bitmap format
01021 /// </summary>
01022         PBM,
01023 
01024 /// <summary>
01025 /// Portable network graphics format
01026 /// </summary>
01027         PNG,
01028 
01029 /// <summary>
01030 /// Truevision graphics adapter format
01031 /// </summary>
01032         TGA,
01033 
01034 /// <summary>
01035 /// Tag image file format
01036 /// </summary>
01037         TIFF,
01038 
01039 /// <summary>
01040 /// WebP format
01041 /// </summary>
01042         WebP
01043     }
01044 
01045 /// <summary>
01046 /// Contains methods to render a <see cref="Page"/> to an <see cref="Image"/>.
01047 /// </summary>
01048     public static class ImageSharpContextInterpreter
01049     {
01050 /// <summary>
01051 /// Render the page to an <see cref="Image"/> object.
01052 /// </summary>
01053 /// <param name="page">The <see cref="Page"/> to render.</param>
01054 /// <param name="scale">The scale to be used when rasterising the page.  This will determine the width
      and height of the <see cref="Image"/>.</param>
01055 /// <returns>An <see cref="Image"/> containing the rasterised page.</returns>
01056         public static Image<SixLabors.ImageSharp.PixelFormats.Rgba32> SaveAsImage(this Page page,
      double scale = 1)
01057         {
01058             ImageSharpContext ctx = new ImageSharpContext(page.Width, page.Height, scale,
      page.Background);
01059             page.Graphics.CopyToIGraphicsContext(ctx);
01060             ctx.DisposeAllExceptImage();
01061             return ctx.Image;
01062         }
01063 
01064 
01065 /// <summary>
01066 /// Render the page to an image stream.
```

```
01067 /// </summary>
01068 /// <param name="page">The <see cref="Page"/> to render.</param>
01069 /// <param name="imageStream">The <see cref="Stream"/> on which the image data will be
      written.</param>
01070 /// <param name="outputFormat">The format of the image that will be created.</param>
01071 /// <param name="scale">The scale to be used when rasterising the page.  This will determine the width
      and height of the image.</param>
01072         public static void SaveAsImage(this Page page, Stream imageStream, OutputFormats outputFormat,
      double scale = 1)
01073         {
01074             Image image = SaveAsImage(page, scale);
01075 
01076             switch (outputFormat)
01077             {
01078                 case OutputFormats.BMP:
01079                     image.SaveAsBmp(imageStream);
01080                     break;
01081                 case OutputFormats.GIF:
01082                     image.SaveAsGif(imageStream);
01083                     break;
01084                 case OutputFormats.JPEG:
01085                     image.SaveAsJpeg(imageStream);
01086                     break;
01087                 case OutputFormats.PBM:
01088                     image.SaveAsPbm(imageStream);
01089                     break;
01090                 case OutputFormats.PNG:
01091                     image.SaveAsPng(imageStream);
01092                     break;
01093                 case OutputFormats.TGA:
01094                     image.SaveAsTga(imageStream);
01095                     break;
01096                 case OutputFormats.TIFF:
01097                     image.SaveAsTiff(imageStream);
01098                     break;
01099                 case OutputFormats.WebP:
01100                     image.SaveAsWebp(imageStream);
01101                     break;
01102             }
01103 
01104             image.Dispose();
01105         }
01106 
01107 /// <summary>
01108 /// The exception that is raised when the output file format is not specified and the file name does
      not have an extension corresponding to a known file format.
01109 /// </summary>
01110         public class UnknownFormatException :  Exception
01111         {
01112 /// <summary>
01113 /// The extension of the file that does not correspond to any known file format.
01114 /// </summary>
01115             public string Format { get; }
01116 
01117             internal UnknownFormatException(string format) :  base("The extension " + format + " does
      not correspond to any known file format!")
01118             {
01119                 this.Format = format;
01120             }
01121         }
01122 
01123 /// <summary>
01124 /// Render the page to an image file.
01125 /// </summary>
01126 /// <param name="page">The <see cref="Page"/> to render.</param>
01127 /// <param name="fileName">The path of the file where the image will be saved.</param>
01128 /// <param name="outputFormat">The format of the image that will be created.  If this is <see
      langword="null" /> (the default), the format is desumed from the extension of the file.</param>
01129 /// <param name="scale">The scale to be used when rasterising the page.  This will determine the width
      and height of the image.</param>
01130         public static void SaveAsImage(this Page page, string fileName, OutputFormats?  outputFormat =
      null, double scale = 1)
01131         {
01132             OutputFormats actualOutputFormat;
01133 
01134             if (outputFormat.HasValue)
01135             {
01136                 actualOutputFormat = outputFormat.Value;
01137             }
01138             else
01139             {
01140                 string extension = System.IO.Path.GetExtension(fileName).ToLower();
01141 
01142                 switch (extension)
01143                 {
01144                     case ".bmp":
01145                     case ".dib":
```

```
01146                       actualOutputFormat = OutputFormats.BMP;
01147                       break;
01148
01149                   case ".gif":
01150                       actualOutputFormat = OutputFormats.GIF;
01151                       break;
01152
01153                   case ".jpeg":
01154                   case ".jpg":
01155                       actualOutputFormat = OutputFormats.JPEG;
01156                       break;
01157
01158                   case ".pbm":
01159                       actualOutputFormat = OutputFormats.PBM;
01160                       break;
01161
01162                   case ".png":
01163                       actualOutputFormat = OutputFormats.PNG;
01164                       break;
01165
01166                   case ".tga":
01167                   case ".targa":
01168                       actualOutputFormat = OutputFormats.TGA;
01169                       break;
01170
01171                   case ".tif":
01172                   case ".tiff":
01173                       actualOutputFormat = OutputFormats.TIFF;
01174                       break;
01175
01176                   case ".webp":
01177                       actualOutputFormat = OutputFormats.WebP;
01178                       break;
01179
01180                   default:
01181                       throw new UnknownFormatException(extension);
01182               }
01183           }
01184
01185           using (FileStream imageStream = new FileStream(fileName, FileMode.Create))
01186           {
01187               SaveAsImage(page, imageStream, actualOutputFormat, scale);
01188           }
01189       }
01190
01191 /// <summary>
01192 /// Render the page to raw pixel data, in 32bpp RGBA format.
01193 /// </summary>
01194 /// <param name="pag">The <see cref="Page"/> to render.</param>
01195 /// <param name="scale">The scale to be used when rasterising the page.  This will determine the width
      and height of the image.</param>
01196 /// <param name="width">The width of the rendered image.</param>
01197 /// <param name="height">The height of the rendered image.</param>
01198 /// <param name="totalSize">The size in bytes of the raw pixel data.</param>
01199 /// <returns>A <see cref="DisposableIntPtr"/> containing a pointer to the raw pixel data, stored in
      unmanaged memory.  Dispose this object to release the unmanaged memory.</returns>
01200       public static DisposableIntPtr SaveAsRawBytes(this Page pag, out int width, out int height,
      out int totalSize, double scale = 1)
01201       {
01202           Image<SixLabors.ImageSharp.PixelFormats.Rgba32> img = SaveAsImage(pag, scale);
01203
01204           int stride = img.Width * 4;
01205           int size = stride * img.Height;
01206
01207           IntPtr tbr = System.Runtime.InteropServices.Marshal.AllocHGlobal(size);
01208           GC.AddMemoryPressure(size);
01209
01210           IntPtr pointer = tbr;
01211
01212           unsafe
01213           {
01214               for (int y = 0; y < img.Height; y++)
01215               {
01216                   Memory<SixLabors.ImageSharp.PixelFormats.Rgba32> row =
      img.DangerousGetPixelRowMemory(y);
01217
01218                   Span<SixLabors.ImageSharp.PixelFormats.Rgba32> newRow = new
      Span<SixLabors.ImageSharp.PixelFormats.Rgba32>(pointer.ToPointer(), row.Length);
01219                   row.Span.CopyTo(newRow);
01220
01221                   pointer = IntPtr.Add(pointer, stride);
01222               }
01223           }
01224
01225           width = img.Width;
01226           height = img.Height;
01227           totalSize = size;
```

```
01228
01229             img.Dispose();
01230
01231             return new DisposableIntPtr(tbr);
01232         }
01233
01234
01235 /// <summary>
01236 /// Return the page to raw pixel data, in 32bpp RGBA format.
01237 /// </summary>
01238 /// <param name="pag">The <see cref="Page"/> to render.</param>
01239 /// <param name="scale">The scale to be used when rasterising the page.  This will determine the width
      and height of the image.</param>
01240 /// <param name="width">The width of the rendered image.</param>
01241 /// <param name="height">The height of the rendered image.</param>
01242 /// <returns>A byte array containing the raw pixel data.</returns>
01243         public static byte[] SaveAsRawBytes(this Page pag, out int width, out int height, double scale
      = 1)
01244         {
01245             Image<SixLabors.ImageSharp.PixelFormats.Rgba32> img = SaveAsImage(pag, scale);
01246
01247             int stride = img.Width * 4;
01248             int size = stride * img.Height;
01249
01250             byte[] tbr = new byte[size];
01251
01252             unsafe
01253             {
01254                 for (int y = 0; y < img.Height; y++)
01255                 {
01256                     Memory<SixLabors.ImageSharp.PixelFormats.Rgba32> row =
      img.DangerousGetPixelRowMemory(y);
01257
01258                     Span<byte> bytes =
      System.Runtime.InteropServices.MemoryMarshal.Cast<SixLabors.ImageSharp.PixelFormats.Rgba32,
      byte>(row.Span);
01259                     Span<byte> newRow = new Span<byte>(tbr, y * stride, stride);
01260                     bytes.CopyTo(newRow);
01261                 }
01262             }
01263
01264             width = img.Width;
01265             height = img.Height;
01266
01267             img.Dispose();
01268
01269             return tbr;
01270         }
01271
01272 /// <summary>
01273 /// Rasterise a region of a <see cref="Graphics"/> object.
01274 /// </summary>
01275 /// <param name="graphics">The <see cref="Graphics"/> object that will be rasterised.</param>
01276 /// <param name="region">The region of the <paramref name="graphics"/> that will be
      rasterised.</param>
01277 /// <param name="scale">The scale at which the image will be rendered.</param>
01278 /// <param name="interpolate">Whether the resulting image should be interpolated or not when it is
      drawn on another <see cref="Graphics"/> surface.</param>
01279 /// <returns>A <see cref="RasterImage"/> containing the rasterised graphics.</returns>
01280         public static RasterImage Rasterise(this Graphics graphics, Rectangle region, double scale,
      bool interpolate)
01281         {
01282             Page pag = new Page(1, 1);
01283             pag.Graphics.DrawGraphics(0, 0, graphics);
01284             pag.Crop(region.Location, region.Size);
01285
01286             Image<SixLabors.ImageSharp.PixelFormats.Rgba32> img = SaveAsImage(pag, scale);
01287
01288             int stride = img.Width * 4;
01289             int size = stride * img.Height;
01290
01291             IntPtr tbr = System.Runtime.InteropServices.Marshal.AllocHGlobal(size);
01292             GC.AddMemoryPressure(size);
01293
01294             IntPtr pointer = tbr;
01295
01296             unsafe
01297             {
01298                 for (int y = 0; y < img.Height; y++)
01299                 {
01300                     Memory<SixLabors.ImageSharp.PixelFormats.Rgba32> row =
      img.DangerousGetPixelRowMemory(y);
01301
01302                     Span<SixLabors.ImageSharp.PixelFormats.Rgba32> newRow = new
      Span<SixLabors.ImageSharp.PixelFormats.Rgba32>(pointer.ToPointer(), row.Length);
01303                     row.Span.CopyTo(newRow);
01304
```

```
01305                       pointer = IntPtr.Add(pointer, stride);
01306                   }
01307               }
01308
01309               int width = img.Width;
01310               int height = img.Height;
01311
01312               img.Dispose();
01313
01314               DisposableIntPtr disp = new DisposableIntPtr(tbr);
01315
01316               return new RasterImage(ref disp, width, height, true, interpolate);
01317           }
01318
01319 /// <summary>
01320 /// Saves the animation to an animated GIF.
01321 /// </summary>
01322 /// <param name="animation">The animation to export.</param>
01323 /// <param name="scale">The scale at which the animation will be rendered.</param>
01324 /// <param name="frameRate">The target frame rate of the animation, in frames-per-second (fps).  This
      is capped by the animated GIF specification at 50 fps.</param>
01325 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the
      animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note
      that this does not affect the frame rate of the animation.</param>
01326 /// <param name="colorTableMode">Determines whether a single colour table should be used for the whole
      image, or if a different colour table should be used for each frame.</param>
01327 /// <returns>An <see cref="Image"/> containing the animated GIF.</returns>
01328           public static Image SaveAsAnimatedGIF(this Animation animation, double scale = 1, double
      frameRate = 50, double durationScaling = 1, GifColorTableMode colorTableMode =
      GifColorTableMode.Local)
01329           {
01330               frameRate = Math.Min(frameRate, 50);
01331
01332               Image gifAnimation = new
      Image<SixLabors.ImageSharp.PixelFormats.Rgba32>((int)(animation.Width * scale), (int)(animation.Height
      * scale));
01333
01334               int frames = (int)Math.Ceiling(animation.Duration * frameRate * durationScaling / 1000);
01335
01336               double accumulatedDelay = 0;
01337
01338               for (int i = 0; i < frames; i++)
01339               {
01340                   double frameTime = i / frameRate / durationScaling * 1000;
01341                   double frameDuration = Math.Min(animation.Duration - i / durationScaling / frameRate *
      1000, 1000 / frameRate / durationScaling);
01342
01343                   int gifFrameDuration = (int)Math.Round(frameDuration * durationScaling * 0.1 +
      accumulatedDelay);
01344
01345                   if (gifFrameDuration < 2)
01346                   {
01347                       accumulatedDelay += frameDuration * durationScaling * 0.1;
01348                   }
01349                   else
01350                   {
01351                       accumulatedDelay += frameDuration * durationScaling * 0.1 - gifFrameDuration;
01352
01353                       Page pag = animation.GetFrameAtAbsolute(frameTime);
01354
01355                       Image frame = pag.SaveAsImage(scale);
01356
01357                       GifFrameMetadata frameMetadata =
      frame.Frames[0].Metadata.GetFormatMetadata(GifFormat.Instance);
01358                       frameMetadata.FrameDelay = gifFrameDuration;
01359                       frameMetadata.DisposalMethod = GifDisposalMethod.RestoreToBackground;
01360                       gifAnimation.Frames.AddFrame(frame.Frames[0]);
01361                   }
01362               }
01363
01364               if ((int)accumulatedDelay > 0)
01365               {
01366                   gifAnimation.Frames[gifAnimation.Frames.Count -
      1].Metadata.GetFormatMetadata(GifFormat.Instance).FrameDelay += (int)accumulatedDelay;
01367               }
01368
01369               gifAnimation.Frames.RemoveFrame(0);
01370
01371               GifMetadata metadata = gifAnimation.Metadata.GetFormatMetadata(GifFormat.Instance);
01372               metadata.ColorTableMode = colorTableMode;
01373               metadata.RepeatCount = (ushort)animation.RepeatCount;
01374
01375               return gifAnimation;
01376           }
01377
01378 /// <summary>
01379 /// Saves the animation to an animated GIF stream.
```

```
01380 /// </summary>
01381 /// <param name="animation">The animation to export.</param>
01382 /// <param name="imageStream">The stream on which the animated GIF will be written.</param>
01383 /// <param name="scale">The scale at which the animation will be rendered.</param>
01384 /// <param name="frameRate">The target frame rate of the animation, in frames-per-second (fps).  This
      is capped by the animated GIF specification at 50 fps.</param>
01385 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the
      animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note
      that this does not affect the frame rate of the animation.</param>
01386 /// <param name="colorTableMode">Determines whether a single colour table should be used for the whole
      image, or if a different colour table should be used for each frame.</param>
01387         public static void SaveAsAnimatedGIF(this Animation animation, Stream imageStream, double
      scale = 1, double frameRate = 50, double durationScaling = 1, GifColorTableMode colorTableMode =
      GifColorTableMode.Local)
01388         {
01389             Image gifAnimation = SaveAsAnimatedGIF(animation, scale, frameRate, durationScaling,
      colorTableMode);
01390
01391             gifAnimation.SaveAsGif(imageStream);
01392         }
01393
01394 /// <summary>
01395 /// Saves the animation to an animated GIF file.
01396 /// </summary>
01397 /// <param name="animation">The animation to export.</param>
01398 /// <param name="fileName">The output file to create.</param>
01399 /// <param name="scale">The scale at which the animation will be rendered.</param>
01400 /// <param name="frameRate">The target frame rate of the animation, in frames-per-second (fps).  This
      is capped by the animated GIF specification at 50 fps.</param>
01401 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the
      animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note
      that this does not affect the frame rate of the animation.</param>
01402 /// <param name="colorTableMode">Determines whether a single colour table should be used for the whole
      image, or if a different colour table should be used for each frame.</param>
01403         public static void SaveAsAnimatedGIF(this Animation animation, string fileName, double scale =
      1, double frameRate = 50, double durationScaling = 1, GifColorTableMode colorTableMode =
      GifColorTableMode.Local)
01404         {
01405             using (FileStream fs = File.Create(fileName))
01406             {
01407                 SaveAsAnimatedGIF(animation, fs, scale, frameRate, durationScaling, colorTableMode);
01408             }
01409         }
01410
01411 /// <summary>
01412 /// Saves the animation to a stream in animated PNG format.
01413 /// </summary>
01414 /// <param name="animation">The animation to export.</param>
01415 /// <param name="imageStream">The stream on which the animated PNG will be written.</param>
01416 /// <param name="scale">The scale at which the animation will be rendered.</param>
01417 /// <param name="frameRate">The target frame rate of the animation, in frames-per-second (fps).  This
      is capped by the animated PNG specification at 90 fps.</param>
01418 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the
      animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note
      that this does not affect the frame rate of the animation.</param>
01419 /// <param name="interframeCompression">The kind of compression that will be used to reduce file size.
      Note that if the animation has a transparent background, no compression can be performed, and the
      value of this parameter is ignored.</param>
01420         public static void SaveAsAnimatedPNG(this Animation animation, Stream imageStream, double
      scale = 1, double frameRate = 60, double durationScaling = 1, AnimatedPNG.InterframeCompression
      interframeCompression = AnimatedPNG.InterframeCompression.First)
01421         {
01422             frameRate = Math.Min(frameRate, 90);
01423
01424             int frames = (int)Math.Ceiling(animation.Duration * frameRate * durationScaling / 1000);
01425
01426             Stream fs = imageStream;
01427
01428             AnimatedPNG.CompressedFrame[] compressedFrames = new AnimatedPNG.CompressedFrame[frames];
01429
01430             int width = (int)(animation.Width * scale);
01431             int height = (int)(animation.Height * scale);
01432
01433             if (animation.Background.A < 1 || interframeCompression ==
      AnimatedPNG.InterframeCompression.None)
01434             {
01435                 Parallel.For(0, frames, i =>
01436                 {
01437                     double frameTime = i / frameRate / durationScaling * 1000;
01438
01439                     double frameDuration = Math.Min((animation.Duration - frameTime) *
      durationScaling, 1000 / frameRate);
01440
01441                     Page pag = animation.GetFrameAtAbsolute(frameTime);
01442
01443                     using (DisposableIntPtr rawFrame = pag.SaveAsRawBytes(out _, out _, out _, scale))
01444                     {
```

```
01445                         compressedFrames[i] = new AnimatedPNG.CompressedFrame(rawFrame, width, height,
      true, frameDuration);
01446                     }
01447                 });
01448             }
01449             else if (interframeCompression == AnimatedPNG.InterframeCompression.First)
01450             {
01451                 DisposableIntPtr firstFrame = animation.GetFrameAtAbsolute(0).SaveAsRawBytes(out _,
      out _, out _, scale);
01452                 compressedFrames[0] = new AnimatedPNG.CompressedFrame(firstFrame, width, height, true,
      Math.Min(animation.Duration * durationScaling, 1000 / frameRate));
01453 
01454                 Parallel.For(1, frames, i =>
01455                 {
01456                     double frameTime = i / frameRate / durationScaling * 1000;
01457 
01458                     double frameDuration = Math.Min((animation.Duration - frameTime) *
      durationScaling, 1000 / frameRate);
01459 
01460                     Page pag = animation.GetFrameAtAbsolute(frameTime);
01461 
01462                     using (DisposableIntPtr rawFrame = pag.SaveAsRawBytes(out _, out _, out _, scale))
01463                     {
01464                         compressedFrames[i] = new AnimatedPNG.CompressedFrame(rawFrame, firstFrame,
      true, width, height, true, frameDuration);
01465                     }
01466                 });
01467 
01468                 firstFrame.Dispose();
01469             }
01470             else if (interframeCompression == AnimatedPNG.InterframeCompression.Previous)
01471             {
01472                 DisposableIntPtr[] rawFrames = new DisposableIntPtr[frames];
01473 
01474                 Parallel.For(0, frames, i =>
01475                 {
01476                     double frameTime = i / frameRate / durationScaling * 1000;
01477                     Page pag = animation.GetFrameAtAbsolute(frameTime);
01478                     rawFrames[i] = pag.SaveAsRawBytes(out _, out _, out _, scale);
01479                 });
01480 
01481                 compressedFrames[0] = new AnimatedPNG.CompressedFrame(rawFrames[0], width, height,
      true, Math.Min(animation.Duration * durationScaling, 1000 / frameRate));
01482 
01483                 Parallel.For(1, frames, i =>
01484                 {
01485                     double frameTime = i / frameRate / durationScaling * 1000;
01486                     double frameDuration = Math.Min((animation.Duration - frameTime) *
      durationScaling, 1000 / frameRate);
01487                     compressedFrames[i] = new AnimatedPNG.CompressedFrame(rawFrames[i], rawFrames[i -
      1], false, width, height, true, frameDuration);
01488                 });
01489             }
01490 
01491             AnimatedPNG.Create(fs, (int)(animation.Width * scale), (int)(animation.Height * scale),
      true, compressedFrames, animation.RepeatCount);
01492 
01493             for (int i = 0; i < compressedFrames.Length; i++)
01494             {
01495                 compressedFrames[i].Dispose();
01496             }
01497         }
01498 
01499 /// <summary>
01500 /// Saves the animation to an animated PNG file.
01501 /// </summary>
01502 /// <param name="animation">The animation to export.</param>
01503 /// <param name="fileName">The output file to create.</param>
01504 /// <param name="scale">The scale at which the animation will be rendered.</param>
01505 /// <param name="frameRate">The target frame rate of the animation, in frames-per-second (fps).  This
      is capped by the animated PNG specification at 90 fps.</param>
01506 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the
      animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note
      that this does not affect the frame rate of the animation.</param>
01507 /// <param name="interframeCompression">The kind of compression that will be used to reduce file size.
      Note that if the animation has a transparent background, no compression can be performed, and the
      value of this parameter is ignored.</param>
01508         public static void SaveAsAnimatedPNG(this Animation animation, string fileName, double scale =
      1, double frameRate = 60, double durationScaling = 1, AnimatedPNG.InterframeCompression
      interframeCompression = AnimatedPNG.InterframeCompression.First)
01509         {
01510             using (FileStream fs = File.Create(fileName))
01511             {
01512                 SaveAsAnimatedPNG(animation, fs, scale, frameRate, durationScaling,
      interframeCompression);
01513             }
01514         }
```

```
01515     }
01516 }
```

8.60 Raster.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020  Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Affero General Public License as
00007 published by the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012 GNU Affero General Public License for more details.
00013
00014 You should have received a copy of the GNU Affero General Public License
00015 along with this program.    If not, see <http://www.gnu.org/licenses/>
00016 */
00017
00018 using MuPDFCore;
00019 using System;
00020 using System.IO;
00021 using System.Runtime.InteropServices;
00022 using System.Threading.Tasks;
00023 using VectSharp.PDF;
00024
00025 namespace VectSharp.Raster
00026 {
00027 /// <summary>
00028 /// Contains methods to render a page to a PNG image.
00029 /// </summary>
00030     public static class Raster
00031     {
00032
00033 /// <summary>
00034 /// Render the page to a PNG file.
00035 /// </summary>
00036 /// <param name="page">The <see cref="Page"/> to render.</param>
00037 /// <param name="fileName">The full path to the file to save.  If it exists, it will be
      overwritten.</param>
00038 /// <param name="scale">The scale to be used when rasterising the page.  This will determine the width
      and height of the image file.</param>
00039         public static void SaveAsPNG(this Page page, string fileName, double scale = 1)
00040         {
00041             Document doc = new Document();
00042             doc.Pages.Add(page);
00043
00044             MemoryStream ms = new MemoryStream();
00045             doc.SaveAsPDF(ms, textOption:  PDFContextInterpreter.TextOptions.ConvertIntoPaths,
      filterOption:  new
      PDFContextInterpreter.FilterOption(PDFContextInterpreter.FilterOption.FilterOperations.RasteriseAll,
      scale, true));
00046
00047             using (MuPDFContext context = new MuPDFContext())
00048             {
00049                 using (MuPDFDocument muDoc = new MuPDFDocument(context, ref ms, InputFileTypes.PDF))
00050                 {
00051                     muDoc.SaveImage(0, scale, MuPDFCore.PixelFormats.RGBA, fileName,
      RasterOutputFileTypes.PNG);
00052                 }
00053             }
00054
00055             ms.Dispose();
00056         }
00057
00058 /// <summary>
00059 /// Render the page to a PNG stream.
00060 /// </summary>
00061 /// <param name="page">The <see cref="Page"/> to render.</param>
00062 /// <param name="stream">The stream to which the PNG data will be written.</param>
00063 /// <param name="scale">The scale to be used when rasterising the page.  This will determine the width
      and height of the image file.</param>
00064         public static void SaveAsPNG(this Page page, Stream stream, double scale = 1)
00065         {
00066             Document doc = new Document();
00067             doc.Pages.Add(page);
00068
00069             MemoryStream ms = new MemoryStream();
00070             doc.SaveAsPDF(ms, textOption:  PDFContextInterpreter.TextOptions.ConvertIntoPaths,
      filterOption:  new
```

```
      PDFContextInterpreter.FilterOption(PDFContextInterpreter.FilterOption.FilterOperations.RasteriseAll,
      scale, true));
00071
00072             using (MuPDFContext context = new MuPDFContext())
00073             {
00074                 using (MuPDFDocument muDoc = new MuPDFDocument(context, ref ms, InputFileTypes.PDF))
00075                 {
00076                     muDoc.WriteImage(0, scale, MuPDFCore.PixelFormats.RGBA, stream,
      RasterOutputFileTypes.PNG);
00077                 }
00078             }
00079
00080             ms.Dispose();
00081         }
00082
00083 /// <summary>
00084 /// Rasterise a region of a <see cref="Graphics"/> object.
00085 /// </summary>
00086 /// <param name="graphics">The <see cref="Graphics"/> object that will be rasterised.</param>
00087 /// <param name="region">The region of the <paramref name="graphics"/> that will be
      rasterised.</param>
00088 /// <param name="scale">The scale at which the image will be rendered.</param>
00089 /// <param name="interpolate">Whether the resulting image should be interpolated or not when it is
      drawn on another <see cref="Graphics"/> surface.</param>
00090 /// <returns>A <see cref="RasterImage"/> containing the rasterised graphics.</returns>
00091         public static RasterImage Rasterise(this Graphics graphics, Rectangle region, double scale,
      bool interpolate)
00092         {
00093             Page pag = new Page(1, 1);
00094             pag.Graphics.DrawGraphics(0, 0, graphics);
00095             pag.Crop(region.Location, region.Size);
00096
00097             Document doc = new Document();
00098             doc.Pages.Add(pag);
00099
00100             MemoryStream ms = new MemoryStream();
00101             doc.SaveAsPDF(ms, textOption:  PDFContextInterpreter.TextOptions.ConvertIntoPaths,
      filterOption:  new
      PDFContextInterpreter.FilterOption(PDFContextInterpreter.FilterOption.FilterOperations.RasteriseAll,
      scale, true));
00102
00103             IntPtr imageDataAddress;
00104
00105             int width;
00106             int height;
00107
00108             using (MuPDFContext context = new MuPDFContext())
00109             {
00110                 using (MuPDFDocument muDoc = new MuPDFDocument(context, ref ms, InputFileTypes.PDF))
00111                 {
00112                     int imageSize = muDoc.GetRenderedSize(0, scale, MuPDFCore.PixelFormats.RGBA);
00113
00114                     RoundedRectangle roundedBounds = muDoc.Pages[0].Bounds.Round(scale);
00115
00116                     width = roundedBounds.Width;
00117                     height = roundedBounds.Height;
00118
00119
00120                     imageDataAddress = Marshal.AllocHGlobal(imageSize);
00121                     GC.AddMemoryPressure(imageSize);
00122
00123                     muDoc.Render(0, scale, MuPDFCore.PixelFormats.RGBA, imageDataAddress);
00124                 }
00125             }
00126
00127             ms.Dispose();
00128
00129             DisposableIntPtr disposableAddress = new DisposableIntPtr(imageDataAddress);
00130
00131             return new RasterImage(ref disposableAddress, width, height, true, interpolate);
00132         }
00133
00134 /// <summary>
00135 /// Render the page to raw pixel data, in 32bpp RGBA format.
00136 /// </summary>
00137 /// <param name="pag">The <see cref="Page"/> to render.</param>
00138 /// <param name="scale">The scale to be used when rasterising the page.  This will determine the width
      and height of the image.</param>
00139 /// <param name="width">The width of the rendered image.</param>
00140 /// <param name="height">The height of the rendered image.</param>
00141 /// <param name="totalSize">The size in bytes of the raw pixel data.</param>
00142 /// <returns>A <see cref="DisposableIntPtr"/> containing a pointer to the raw pixel data, stored in
      unmanaged memory.  Dispose this object to release the unmanaged memory.</returns>
00143         public static DisposableIntPtr SaveAsRawBytes(this Page pag, out int width, out int height,
      out int totalSize, double scale = 1)
00144         {
00145             Document doc = new Document();
```

```
00146             doc.Pages.Add(pag);
00147
00148             MemoryStream ms = new MemoryStream();
00149             doc.SaveAsPDF(ms, textOption:  PDFContextInterpreter.TextOptions.ConvertIntoPaths,
      filterOption:  new
      PDFContextInterpreter.FilterOption(PDFContextInterpreter.FilterOption.FilterOperations.RasteriseAll,
      scale, true));
00150
00151             IntPtr renderedPage;
00152
00153             using (MuPDFContext context = new MuPDFContext())
00154             {
00155                 using (MuPDFDocument muDoc = new MuPDFDocument(context, ref ms, InputFileTypes.PDF))
00156                 {
00157                     totalSize = muDoc.GetRenderedSize(0, scale, MuPDFCore.PixelFormats.RGBA);
00158
00159                     renderedPage = Marshal.AllocHGlobal(totalSize);
00160
00161                     RoundedRectangle bounds = muDoc.Pages[0].Bounds.Round(scale);
00162
00163                     width = bounds.Width;
00164                     height = bounds.Height;
00165
00166                     muDoc.Render(0, scale, MuPDFCore.PixelFormats.RGBA, renderedPage);
00167                 }
00168             }
00169
00170             ms.Dispose();
00171
00172             return new DisposableIntPtr(renderedPage);
00173         }
00174
00175 /// <summary>
00176 /// Saves the animation to a stream in animated PNG format.
00177 /// </summary>
00178 /// <param name="animation">The animation to export.</param>
00179 /// <param name="imageStream">The stream on which the animated PNG will be written.</param>
00180 /// <param name="scale">The scale at which the animation will be rendered.</param>
00181 /// <param name="frameRate">The target frame rate of the animation, in frames-per-second (fps).  This
      is capped by the animated PNG specification at 90 fps.</param>
00182 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the
      animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note
      that this does not affect the frame rate of the animation.</param>
00183 /// <param name="interframeCompression">The kind of compression that will be used to reduce file size.
      Note that if the animation has a transparent background, no compression can be performed, and the
      value of this parameter is ignored.</param>
00184         public static void SaveAsAnimatedPNG(this Animation animation, Stream imageStream, double
      scale = 1, double frameRate = 60, double durationScaling = 1, AnimatedPNG.InterframeCompression
      interframeCompression = AnimatedPNG.InterframeCompression.First)
00185         {
00186             frameRate = Math.Min(frameRate, 90);
00187
00188             int frames = (int)Math.Ceiling(animation.Duration * frameRate * durationScaling / 1000);
00189
00190             Stream fs = imageStream;
00191
00192             AnimatedPNG.CompressedFrame[] compressedFrames = new AnimatedPNG.CompressedFrame[frames];
00193
00194             int width = (int)(animation.Width * scale);
00195             int height = (int)(animation.Height * scale);
00196
00197             if (animation.Background.A < 1 || interframeCompression ==
      AnimatedPNG.InterframeCompression.None)
00198             {
00199                 Parallel.For(0, frames, i =>
00200                 {
00201                     double frameTime = i / frameRate / durationScaling * 1000;
00202
00203                     double frameDuration = Math.Min((animation.Duration - frameTime) *
      durationScaling, 1000 / frameRate);
00204
00205                     Page pag = animation.GetFrameAtAbsolute(frameTime);
00206
00207                     using (DisposableIntPtr rawFrame = pag.SaveAsRawBytes(out _, out _, out _, scale))
00208                     {
00209                         compressedFrames[i] = new AnimatedPNG.CompressedFrame(rawFrame, width, height,
      true, frameDuration);
00210                     }
00211                 });
00212             }
00213             else if (interframeCompression == AnimatedPNG.InterframeCompression.First)
00214             {
00215                 DisposableIntPtr firstFrame = animation.GetFrameAtAbsolute(0).SaveAsRawBytes(out _,
      out _, out _, scale);
00216                 compressedFrames[0] = new AnimatedPNG.CompressedFrame(firstFrame, width, height, true,
      Math.Min(animation.Duration * durationScaling, 1000 / frameRate));
00217
```

```
00218                 Parallel.For(1, frames, i =>
00219                 {
00220                     double frameTime = i / frameRate / durationScaling * 1000;
00221 
00222                     double frameDuration = Math.Min((animation.Duration - frameTime) *
      durationScaling, 1000 / frameRate);
00223 
00224                     Page pag = animation.GetFrameAtAbsolute(frameTime);
00225 
00226                     using (DisposableIntPtr rawFrame = pag.SaveAsRawBytes(out _, out _, out _, scale))
00227                     {
00228                         compressedFrames[i] = new AnimatedPNG.CompressedFrame(rawFrame, firstFrame,
      true, width, height, true, frameDuration);
00229                     }
00230                 });
00231 
00232                 firstFrame.Dispose();
00233             }
00234             else if (interframeCompression == AnimatedPNG.InterframeCompression.Previous)
00235             {
00236                 DisposableIntPtr[] rawFrames = new DisposableIntPtr[frames];
00237 
00238                 Parallel.For(0, frames, i =>
00239                 {
00240                     double frameTime = i / frameRate / durationScaling * 1000;
00241                     Page pag = animation.GetFrameAtAbsolute(frameTime);
00242                     rawFrames[i] = pag.SaveAsRawBytes(out _, out _, out _, scale);
00243                 });
00244 
00245                 compressedFrames[0] = new AnimatedPNG.CompressedFrame(rawFrames[0], width, height,
      true, Math.Min(animation.Duration * durationScaling, 1000 / frameRate));
00246 
00247                 Parallel.For(1, frames, i =>
00248                 {
00249                     double frameTime = i / frameRate / durationScaling * 1000;
00250                     double frameDuration = Math.Min((animation.Duration - frameTime) *
      durationScaling, 1000 / frameRate);
00251                     compressedFrames[i] = new AnimatedPNG.CompressedFrame(rawFrames[i], rawFrames[i -
      1], false, width, height, true, frameDuration);
00252                 });
00253             }
00254 
00255             AnimatedPNG.Create(fs, (int)(animation.Width * scale), (int)(animation.Height * scale),
      true, compressedFrames, animation.RepeatCount);
00256         }
00257 
00258 /// <summary>
00259 /// Saves the animation to an animated PNG file.
00260 /// </summary>
00261 /// <param name="animation">The animation to export.</param>
00262 /// <param name="fileName">The output file to create.</param>
00263 /// <param name="scale">The scale at which the animation will be rendered.</param>
00264 /// <param name="frameRate">The target frame rate of the animation, in frames-per-second (fps).  This
      is capped by the animated PNG specification at 90 fps.</param>
00265 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the
      animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note
      that this does not affect the frame rate of the animation.</param>
00266 /// <param name="interframeCompression">The kind of compression that will be used to reduce file size.
      Note that if the animation has a transparent background, no compression can be performed, and the
      value of this parameter is ignored.</param>
00267         public static void SaveAsAnimatedPNG(this Animation animation, string fileName, double scale =
      1, double frameRate = 60, double durationScaling = 1, AnimatedPNG.InterframeCompression
      interframeCompression = AnimatedPNG.InterframeCompression.First)
00268         {
00269             using (FileStream fs = File.Create(fileName))
00270             {
00271                 SaveAsAnimatedPNG(animation, fs, scale, frameRate, durationScaling,
      interframeCompression);
00272             }
00273         }
00274 
00275 
00276 /// <summary>
00277 /// Render the page to a JPEG file.
00278 /// </summary>
00279 /// <param name="page">The <see cref="Page"/> to render.</param>
00280 /// <param name="fileName">The full path to the file to save.  If it exists, it will be
      overwritten.</param>
00281 /// <param name="scale">The scale to be used when rasterising the page.  This will determine the width
      and height of the image file.</param>
00282 /// <param name="quality">The quality of the JPEG output (ranging from 0 to 100).</param>
00283         public static void SaveAsJPEG(this Page page, string fileName, double scale = 1, int quality =
      90)
00284         {
00285             Document doc = new Document();
00286             doc.Pages.Add(page);
00287 
```

```
00288             MemoryStream ms = new MemoryStream();
00289             doc.SaveAsPDF(ms, textOption:  PDFContextInterpreter.TextOptions.ConvertIntoPaths,
      filterOption:  new
      PDFContextInterpreter.FilterOption(PDFContextInterpreter.FilterOption.FilterOperations.RasteriseAll,
      scale, true));
00290
00291             using (MuPDFContext context = new MuPDFContext())
00292             {
00293                 using (MuPDFDocument muDoc = new MuPDFDocument(context, ref ms, InputFileTypes.PDF))
00294                 {
00295                     muDoc.SaveImageAsJPEG(0, scale, fileName, quality);
00296                 }
00297             }
00298
00299             ms.Dispose();
00300         }
00301     }
00302 }
```

8.61 SVGContext.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.IO;
00021 using System.Linq;
00022 using System.Reflection;
00023 using System.Runtime.InteropServices.ComTypes;
00024 using System.Text;
00025 using System.Text.RegularExpressions;
00026 using System.Threading.Tasks;
00027 using System.Xml;
00028 using VectSharp.Filters;
00029
00030 namespace VectSharp.SVG
00031 {
00032     internal static class MatrixUtils
00033     {
00034         public static double[] Multiply(double[,] matrix, double[] vector)
00035         {
00036             double[] tbr = new double[2];
00037
00038             tbr[0] = matrix[0, 0] * vector[0] + matrix[0, 1] * vector[1] + matrix[0, 2];
00039             tbr[1] = matrix[1, 0] * vector[0] + matrix[1, 1] * vector[1] + matrix[1, 2];
00040
00041             return tbr;
00042         }
00043
00044         public static Point Multiply(double[,] matrix, Point vector)
00045         {
00046             double[] tbr = new double[2];
00047
00048             tbr[0] = matrix[0, 0] * vector.X + matrix[0, 1] * vector.Y + matrix[0, 2];
00049             tbr[1] = matrix[1, 0] * vector.X + matrix[1, 1] * vector.Y + matrix[1, 2];
00050
00051             return new Point(tbr[0], tbr[1]);
00052         }
00053
00054         public static double[,] Multiply(double[,] matrix1, double[,] matrix2)
00055         {
00056             double[,] tbr = new double[3, 3];
00057
00058             for (int i = 0; i < 3; i++)
00059             {
00060                 for (int j = 0; j < 3; j++)
00061                 {
00062                     for (int k = 0; k < 3; k++)
```

```
00063                     {
00064                         tbr[i, j] += matrix1[i, k] * matrix2[k, j];
00065                     }
00066                 }
00067             }
00068 
00069             return tbr;
00070         }
00071 
00072         public static double[,] Rotate(double[,] matrix, double angle)
00073         {
00074             double[,] rotationMatrix = new double[3, 3];
00075             rotationMatrix[0, 0] = Math.Cos(angle);
00076             rotationMatrix[0, 1] = -Math.Sin(angle);
00077             rotationMatrix[1, 0] = Math.Sin(angle);
00078             rotationMatrix[1, 1] = Math.Cos(angle);
00079             rotationMatrix[2, 2] = 1;
00080 
00081             return Multiply(matrix, rotationMatrix);
00082         }
00083 
00084         public static double[,] Translate(double[,] matrix, double x, double y)
00085         {
00086             double[,] translationMatrix = new double[3, 3];
00087             translationMatrix[0, 0] = 1;
00088             translationMatrix[0, 2] = x;
00089             translationMatrix[1, 1] = 1;
00090             translationMatrix[1, 2] = y;
00091             translationMatrix[2, 2] = 1;
00092 
00093             return Multiply(matrix, translationMatrix);
00094         }
00095 
00096         public static double[,] Scale(double[,] matrix, double scaleX, double scaleY)
00097         {
00098             double[,] scaleMatrix = new double[3, 3];
00099             scaleMatrix[0, 0] = scaleX;
00100             scaleMatrix[1, 1] = scaleY;
00101             scaleMatrix[2, 2] = 1;
00102 
00103             return Multiply(matrix, scaleMatrix);
00104         }
00105 
00106         public static double[,] Identity = new double[,] { { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0, 1 } };
00107 
00108         public static double[,] Invert(double[,] m)
00109         {
00110             double[,] tbr = new double[3, 3];
00111 
00112             tbr[0, 0] = (m[1, 1] * m[2, 2] - m[1, 2] * m[2, 1]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00113             tbr[0, 1] = -(m[0, 1] * m[2, 2] - m[0, 2] * m[2, 1]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00114             tbr[0, 2] = (m[0, 1] * m[1, 2] - m[0, 2] * m[1, 1]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00115             tbr[1, 0] = -(m[1, 0] * m[2, 2] - m[1, 2] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00116             tbr[1, 1] = (m[0, 0] * m[2, 2] - m[0, 2] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00117             tbr[1, 2] = -(m[0, 0] * m[1, 2] - m[0, 2] * m[1, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00118             tbr[2, 0] = (m[1, 0] * m[2, 1] - m[1, 1] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00119             tbr[2, 1] = -(m[0, 0] * m[2, 1] - m[0, 1] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00120             tbr[2, 2] = (m[0, 0] * m[1, 1] - m[0, 1] * m[1, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
00121 
00122             return tbr;
00123         }
00124 
00125         public static Func<Point, Point> GetInverseTransformation(double[,] m)
00126         {
00127             double[,] inverted = Invert(m);
00128 
00129             return x => Multiply(inverted, x);
00130         }
00131     }
```

```
00132
00133     internal class SVGFigure
00134     {
00135         public Point StartPoint { get; set; }
00136         public Point CurrentPoint { get; set; }
00137
00138         public string Data { get; set; }
00139         public int PointCount { get; set; } = 0;
00140     }
00141
00142     internal class SVGPathObject
00143     {
00144         public List<SVGFigure> Figures { get; set; } = new List<SVGFigure>();
00145     }
00146
00147
00148     internal class SVGContext :  IGraphicsContext
00149     {
00150         public Dictionary<string, FontFamily> UsedFontFamilies;
00151         public Dictionary<string, HashSet<char» UsedChars;
00152         public Dictionary<string, RasterImage> UsedRasterImages;
00153
00154
00155         public const string SVGNamespace = "http://www.w3.org/2000/svg";
00156
00157         public double Width { get; }
00158
00159         public double Height { get; }
00160
00161         public Font Font { get; set; }
00162         public TextBaselines TextBaseline { get; set; }
00163
00164         public Brush FillStyle { get; private set; }
00165
00166         public Brush StrokeStyle { get; private set; }
00167
00168         public double LineWidth { get; set; }
00169         public LineCaps LineCap { private get; set; }
00170         public LineJoins LineJoin { private get; set; }
00171
00172         private LineDash _lineDash;
00173
00174         private SVGPathObject currentPath;
00175         private SVGFigure currentFigure;
00176
00177         public XmlDocument Document;
00178         private XmlElement currentElement;
00179
00180         public string Tag { get; set; }
00181         public string TagPrefix { get; }
00182
00183         private bool ReuseGradients { get; }
00184
00185         private double[,] _transform;
00186         private Stack<double[,]> states;
00187
00188         private string _currClipPath;
00189         private Stack<string> clipPaths;
00190
00191         private bool TextToPaths = false;
00192         private SVGContextInterpreter.TextOptions TextOption =
      SVGContextInterpreter.TextOptions.SubsetFonts;
00193
00194         private Dictionary<string, string> linkDestinations;
00195
00196         public XmlElement Definitions { get; }
00197         Dictionary<Brush, string> gradients;
00198         int gradientCount = 0;
00199
00200         private SVGContextInterpreter.FilterOption FilterOption;
00201
00202         private bool UseStyles;
00203         internal Dictionary<string, Dictionary<string, string» Styles;
00204         XmlElement StyleElement;
00205
00206         private object lockObject;
00207
00208         private string GetClass(Dictionary<string, string> style)
00209         {
00210             foreach (KeyValuePair<string, Dictionary<string, string» styleClass in Styles)
00211             {
00212                 if (styleClass.Value.Count == style.Count)
00213                 {
00214                     bool mismatch = false;
00215
00216                     foreach (KeyValuePair<string, string> styleItem in style)
00217                     {
```

```
00218                         if (!styleClass.Value.TryGetValue(styleItem.Key, out string value))
00219                         {
00220                             mismatch = true;
00221                             break;
00222                         }
00223                         else if (value != styleItem.Value)
00224                         {
00225                             mismatch = true;
00226                             break;
00227                         }
00228                     }
00229 
00230                     if (!mismatch)
00231                     {
00232                         return styleClass.Key;
00233                     }
00234                 }
00235             }
00236 
00237             string tbr = "class" + (Styles.Count + 1).ToString();
00238 
00239             Styles.Add(tbr, style);
00240 
00241             return tbr;
00242         }
00243 
00244         public SVGContext(double width, double height, bool textToPaths,
      SVGContextInterpreter.TextOptions textOption, Dictionary<string, string> linkDestinations,
      SVGContextInterpreter.FilterOption filterOption, bool useStyles, string tagPrefix, bool
      reuseGradients, Dictionary<string, RasterImage> usedRasterImages, object lockObject)
00245         {
00246             this.linkDestinations = linkDestinations;
00247 
00248             this.TagPrefix = tagPrefix;
00249 
00250             this.Width = width;
00251             this.Height = height;
00252 
00253             currentPath = new SVGPathObject();
00254             currentFigure = new SVGFigure();
00255 
00256             StrokeStyle = Colour.FromRgba(0, 0, 0, 0);
00257             FillStyle = Colour.FromRgb(0, 0, 0);
00258             LineWidth = 1;
00259 
00260             LineCap = LineCaps.Butt;
00261             LineJoin = LineJoins.Miter;
00262             _lineDash = new LineDash(0, 0, 0);
00263 
00264             Font = new Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica),
      12);
00265 
00266             TextBaseline = TextBaselines.Top;
00267 
00268             _transform = new double[3, 3];
00269 
00270             _transform[0, 0] = 1;
00271             _transform[1, 1] = 1;
00272             _transform[2, 2] = 1;
00273 
00274             states = new Stack<double[,]>();
00275 
00276             clipPaths = new Stack<string>();
00277             clipPaths.Push(null);
00278             _currClipPath = null;
00279 
00280             UsedFontFamilies = new Dictionary<string, FontFamily>();
00281             UsedChars = new Dictionary<string, HashSet<char»();
00282 
00283             Document = new XmlDocument();
00284 
00285             Document.InsertBefore(Document.CreateXmlDeclaration("1.0", "UTF-8", null),
      Document.DocumentElement);
00286 
00287             currentElement = Document.CreateElement(null, "svg", SVGNamespace);
00288             currentElement.SetAttribute("xmlns:xlink", "http://www.w3.org/1999/xlink");
00289             currentElement.SetAttribute("viewBox", "0 0 " +
      width.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      height.ToString(System.Globalization.CultureInfo.InvariantCulture));
00290             currentElement.SetAttribute("version", "1.1");
00291             Document.AppendChild(currentElement);
00292 
00293             Definitions = Document.CreateElement("defs", SVGNamespace);
00294             gradients = new Dictionary<Brush, string>();
00295             currentElement.AppendChild(Definitions);
00296 
00297             this.UseStyles = useStyles;
```

```
00298             StyleElement = Document.CreateElement("style", SVGNamespace);
00299             StyleElement.InnerText = "\t";
00300             currentElement.AppendChild(StyleElement);
00301             Styles = new Dictionary<string, Dictionary<string, string»();
00302 
00303             this.TextToPaths = textToPaths;
00304             this.TextOption = textOption;
00305             this.FilterOption = filterOption;
00306             this.ReuseGradients = reuseGradients;
00307             UsedRasterImages = usedRasterImages;
00308             this.lockObject = lockObject;
00309         }
00310 
00311 
00312         private void PathText(string text, double x, double y)
00313         {
00314             GraphicsPath textPath = new GraphicsPath().AddText(x, y, text, Font, TextBaseline);
00315 
00316             for (int j = 0; j < textPath.Segments.Count; j++)
00317             {
00318                 switch (textPath.Segments[j].Type)
00319                 {
00320                     case VectSharp.SegmentType.Move:
00321                         this.MoveTo(textPath.Segments[j].Point.X, textPath.Segments[j].Point.Y);
00322                         break;
00323                     case VectSharp.SegmentType.Line:
00324                         this.LineTo(textPath.Segments[j].Point.X, textPath.Segments[j].Point.Y);
00325                         break;
00326                     case VectSharp.SegmentType.CubicBezier:
00327                         this.CubicBezierTo(textPath.Segments[j].Points[0].X,
      textPath.Segments[j].Points[0].Y, textPath.Segments[j].Points[1].X, textPath.Segments[j].Points[1].Y,
      textPath.Segments[j].Points[2].X, textPath.Segments[j].Points[2].Y);
00328                         break;
00329                     case VectSharp.SegmentType.Close:
00330                         this.Close();
00331                         break;
00332                 }
00333             }
00334         }
00335 
00336         public void Close()
00337         {
00338             if (currentFigure.PointCount > 0)
00339             {
00340                 currentFigure.Data += "Z ";
00341                 currentPath.Figures.Add(currentFigure);
00342 
00343                 currentFigure = new SVGFigure();
00344             }
00345         }
00346 
00347         public void CubicBezierTo(double p1X, double p1Y, double p2X, double p2Y, double p3X, double
      p3Y)
00348         {
00349             if (currentFigure.PointCount == 0)
00350             {
00351                 currentFigure.StartPoint = new Point(p1X, p1Y);
00352             }
00353 
00354             currentFigure.CurrentPoint = new Point(p3X, p3Y);
00355             currentFigure.Data += "C " +
      p1X.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      p1Y.ToString(System.Globalization.CultureInfo.InvariantCulture) + ", " +
00356                 p2X.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      p2Y.ToString(System.Globalization.CultureInfo.InvariantCulture) + ", " +
00357                 p3X.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      p3Y.ToString(System.Globalization.CultureInfo.InvariantCulture) + " ";
00358             currentFigure.PointCount += 3;
00359         }
00360 
00361         public void Fill(FillRule fillRule)
00362         {
00363             if (currentFigure.PointCount > 0)
00364             {
00365                 currentPath.Figures.Add(currentFigure);
00366             }
00367 
00368             XmlElement currElement = currentElement;
00369 
00370             if (!string.IsNullOrEmpty(_currClipPath))
00371             {
00372                 currentElement = Document.CreateElement("g", SVGNamespace);
00373                 currentElement.SetAttribute("clip-path", _currClipPath);
00374                 currElement.AppendChild(currentElement);
00375             }
00376 
00377             XmlElement path = Document.CreateElement("path", SVGNamespace);
```

```
00378             path.SetAttribute("d", currentPath.Figures.Aggregate("", (a, b) => a + b.Data));
00379
00380             string gradientName = null;
00381
00382             {
00383                 if (FillStyle is LinearGradientBrush linearGradient)
00384                 {
00385                     if (!ReuseGradients || !gradients.TryGetValue(linearGradient, out gradientName))
00386                     {
00387                         if (!string.IsNullOrEmpty(Tag))
00388                         {
00389                             gradientName = this.TagPrefix + "gradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Tag;
00390                         }
00391                         else
00392                         {
00393                             gradientName = this.TagPrefix + "gradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Guid.NewGuid().ToString("N");
00394                         }
00395
00396                         XmlElement gradientElement = linearGradient.ToLinearGradient(Document,
      gradientName);
00397                         this.Definitions.AppendChild(gradientElement);
00398
00399                         if (ReuseGradients)
00400                         {
00401                             gradients.Add(linearGradient, gradientName);
00402                         }
00403
00404                         gradientCount++;
00405                     }
00406                 }
00407                 else if (FillStyle is RadialGradientBrush radialGradient)
00408                 {
00409                     if (!ReuseGradients || !gradients.TryGetValue(radialGradient, out gradientName))
00410                     {
00411                         if (!string.IsNullOrEmpty(Tag))
00412                         {
00413                             gradientName = this.TagPrefix + "gradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Tag;
00414                         }
00415                         else
00416                         {
00417                             gradientName = this.TagPrefix + "gradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Guid.NewGuid().ToString("N");
00418                         }
00419
00420                         XmlElement gradientElement = radialGradient.ToRadialGradient(Document,
      gradientName);
00421                         this.Definitions.AppendChild(gradientElement);
00422
00423                         if (ReuseGradients)
00424                         {
00425                             gradients.Add(radialGradient, gradientName);
00426                         }
00427
00428                         gradientCount++;
00429                     }
00430                 }
00431             }
00432
00433             if (!UseStyles)
00434             {
00435                 path.SetAttribute("stroke", "none");
00436
00437                 switch (fillRule)
00438                 {
00439                     case FillRule.EvenOdd:
00440                         path.SetAttribute("fill-rule", "evenodd");
00441                         break;
00442
00443                     case FillRule.NonZeroWinding:
00444                         path.SetAttribute("fill-rule", "nonzero");
00445                         break;
00446                 }
00447
00448                 if (FillStyle is SolidColourBrush solid)
00449                 {
00450                     path.SetAttribute("fill", solid.Colour.ToCSSString(false));
00451                     path.SetAttribute("fill-opacity",
      solid.A.ToString(System.Globalization.CultureInfo.InvariantCulture));
00452                 }
00453                 else if (FillStyle is LinearGradientBrush linearGradient)
00454                 {
00455                     path.SetAttribute("fill", "url(#" + gradientName + ")");
00456                 }
00457                 else if (FillStyle is RadialGradientBrush radialGradient)
```

```
00458                 {
00459                     path.SetAttribute("fill", "url(#" + gradientName + ")");
00460                 }
00461 
00462                 path.SetAttribute("transform", "matrix(" + _transform[0,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + _transform[1,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
00463                     "," + _transform[0, 1].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
00464                     "," + _transform[0, 2].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
00465             }
00466             else
00467             {
00468                 Dictionary<string, string> style = new Dictionary<string, string>();
00469 
00470                 style.Add("stroke", "none");
00471 
00472                 switch (fillRule)
00473                 {
00474                     case FillRule.EvenOdd:
00475                         style.Add("fill-rule", "evenodd");
00476                         break;
00477 
00478                     case FillRule.NonZeroWinding:
00479                         style.Add("fill-rule", "nonzero");
00480                         break;
00481                 }
00482 
00483                 if (FillStyle is SolidColourBrush solid)
00484                 {
00485                     style.Add("fill", solid.Colour.ToCSSString(false));
00486                     style.Add("fill-opacity",
      solid.A.ToString(System.Globalization.CultureInfo.InvariantCulture));
00487                 }
00488                 else if (FillStyle is LinearGradientBrush linearGradient)
00489                 {
00490                     style.Add("fill", "url(#" + gradientName + ")");
00491                 }
00492                 else if (FillStyle is RadialGradientBrush radialGradient)
00493                 {
00494                     style.Add("fill", "url(#" + gradientName + ")");
00495                 }
00496 
00497                 style.Add("transform", "matrix(" + _transform[0,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + _transform[1,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
00498                     "," + _transform[0, 1].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
00499                     "," + _transform[0, 2].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
00500 
00501                 string className = GetClass(style);
00502 
00503                 path.SetAttribute("class", className);
00504             }
00505 
00506             if (!string.IsNullOrEmpty(Tag))
00507             {
00508                 path.SetAttribute("id", this.TagPrefix + Tag);
00509             }
00510 
00511             if (!string.IsNullOrEmpty(this.Tag) && this.linkDestinations.TryGetValue(this.Tag, out
      string destination) && !string.IsNullOrEmpty(destination))
00512             {
00513                 XmlElement aElement = Document.CreateElement("a", SVGNamespace);
00514                 aElement.SetAttribute("href", destination);
00515                 currentElement.AppendChild(aElement);
00516                 currentElement = aElement;
00517             }
00518 
00519             currentElement.AppendChild(path);
00520 
00521             currentElement = currElement;
00522 
00523             currentPath = new SVGPathObject();
00524             currentFigure = new SVGFigure();
00525 
00526         }
00527 
00528         public void FillText(string text, double x, double y)
00529         {
00530             if ((!TextToPaths && TextOption !=
      SVGContextInterpreter.TextOptions.ConvertIntoPathsUsingGlyphs) || (TextOption ==
      SVGContextInterpreter.TextOptions.ConvertIntoPathsUsingGlyphs && FillStyle is SolidColourBrush))
00531             {
00532                 if (!UsedFontFamilies.ContainsKey(Font.FontFamily.FileName))
```

```
00533                 {
00534                     UsedFontFamilies.Add(Font.FontFamily.FileName, Font.FontFamily);
00535                     UsedChars.Add(Font.FontFamily.FileName, new HashSet<char>());
00536                 }
00537 
00538                 UsedChars[Font.FontFamily.FileName].UnionWith(text);
00539 
00540                 Font.DetailedFontMetrics metrics = Font.MeasureTextAdvanced(text);
00541 
00542                 double[,] currTransform = null;
00543                 double[,] deltaTransform = MatrixUtils.Identity;
00544 
00545                 switch (TextBaseline)
00546                 {
00547                     case TextBaselines.Baseline:
00548                         currTransform = MatrixUtils.Translate(_transform, x - metrics.LeftSideBearing,
      y);
00549                         deltaTransform = MatrixUtils.Translate(deltaTransform, x -
      metrics.LeftSideBearing, y);
00550                         break;
00551                     case TextBaselines.Top:
00552                         currTransform = MatrixUtils.Translate(_transform, x - metrics.LeftSideBearing,
      y + metrics.Top);
00553                         deltaTransform = MatrixUtils.Translate(deltaTransform, x -
      metrics.LeftSideBearing, y + metrics.Top);
00554                         break;
00555                     case TextBaselines.Bottom:
00556                         currTransform = MatrixUtils.Translate(_transform, x - metrics.LeftSideBearing,
      y + metrics.Bottom);
00557                         deltaTransform = MatrixUtils.Translate(deltaTransform, x -
      metrics.LeftSideBearing, y + metrics.Bottom);
00558                         break;
00559                     case TextBaselines.Middle:
00560                         currTransform = MatrixUtils.Translate(_transform, x - metrics.LeftSideBearing,
      y + (metrics.Top + metrics.Bottom) * 0.5);
00561                         deltaTransform = MatrixUtils.Translate(deltaTransform, x -
      metrics.LeftSideBearing, y + (metrics.Top + metrics.Bottom) * 0.5);
00562                         break;
00563                     default:
00564                         currTransform = MatrixUtils.Translate(_transform, x - metrics.LeftSideBearing,
      y);
00565                         deltaTransform = MatrixUtils.Translate(deltaTransform, x -
      metrics.LeftSideBearing, y);
00566                         break;
00567                 }
00568 
00569                 XmlElement currElement = currentElement;
00570 
00571                 if (!string.IsNullOrEmpty(_currClipPath))
00572                 {
00573                     currentElement = Document.CreateElement("g", SVGNamespace);
00574                     currentElement.SetAttribute("clip-path", _currClipPath);
00575                     currElement.AppendChild(currentElement);
00576                 }
00577 
00578                 XmlElement textElement;
00579 
00580                 if (TextOption != SVGContextInterpreter.TextOptions.ConvertIntoPathsUsingGlyphs)
00581                 {
00582                     textElement = Document.CreateElement("text", SVGNamespace);
00583                 }
00584                 else
00585                 {
00586                     textElement = Document.CreateElement("g", SVGNamespace);
00587                 }
00588 
00589 
00590                 string gradientName = null;
00591 
00592                 {
00593                     if (FillStyle is LinearGradientBrush linearGradient)
00594                     {
00595                         if (!string.IsNullOrEmpty(Tag))
00596                         {
00597                             gradientName = this.TagPrefix + "textGradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Tag;
00598                         }
00599                         else
00600                         {
00601                             gradientName = this.TagPrefix + "textGradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Guid.NewGuid().ToString("N");
00602                         }
00603 
00604                         XmlElement gradientElement = linearGradient.ToLinearGradient(Document,
      gradientName);
00605 
00606                         deltaTransform = MatrixUtils.Invert(deltaTransform);
```

```
00607
00608                       gradientElement.SetAttribute("gradientTransform", "matrix(" +
    deltaTransform[0, 0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," +
    deltaTransform[1, 0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
00609                       "," + deltaTransform[0,
    1].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + deltaTransform[1,
    1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
00610                       "," + deltaTransform[0,
    2].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + deltaTransform[1,
    2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
00611
00612                       this.Definitions.AppendChild(gradientElement);
00613
00614                       gradientCount++;
00615                   }
00616                   else if (FillStyle is RadialGradientBrush radialGradient)
00617                   {
00618                       if (!string.IsNullOrEmpty(Tag))
00619                       {
00620                           gradientName = this.TagPrefix + "textGradient" + (gradientCount +
    1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Tag;
00621                       }
00622                       else
00623                       {
00624                           gradientName = this.TagPrefix + "textGradient" + (gradientCount +
    1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Guid.NewGuid().ToString("N");
00625                       }
00626
00627                       XmlElement gradientElement = radialGradient.ToRadialGradient(Document,
    gradientName);
00628
00629                       deltaTransform = MatrixUtils.Invert(deltaTransform);
00630
00631                       gradientElement.SetAttribute("gradientTransform", "matrix(" +
    deltaTransform[0, 0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," +
    deltaTransform[1, 0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
00632                       "," + deltaTransform[0,
    1].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + deltaTransform[1,
    1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
00633                       "," + deltaTransform[0,
    2].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + deltaTransform[1,
    2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
00634
00635                       this.Definitions.AppendChild(gradientElement);
00636
00637                       gradientCount++;
00638                   }
00639               }
00640
00641               if (TextOption != SVGContextInterpreter.TextOptions.ConvertIntoPathsUsingGlyphs)
00642               {
00643                   textElement.SetAttribute("x", "0");
00644                   textElement.SetAttribute("y", "0");
00645               }
00646
00647               textElement.SetAttribute("transform", "matrix(" + currTransform[0,
    0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + currTransform[1,
    0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
00648                       "," + currTransform[0,
    1].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + currTransform[1,
    1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
00649                       "," + currTransform[0,
    2].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + currTransform[1,
    2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
00650
00651               if (!UseStyles)
00652               {
00653                   textElement.SetAttribute("stroke", "none");
00654
00655                   if (FillStyle is SolidColourBrush solid)
00656                   {
00657                       textElement.SetAttribute("fill", solid.Colour.ToCSSString(false));
00658                       textElement.SetAttribute("fill-opacity",
    solid.A.ToString(System.Globalization.CultureInfo.InvariantCulture));
00659                   }
00660                   else if (FillStyle is LinearGradientBrush linearGradient)
00661                   {
00662                       textElement.SetAttribute("fill", "url(#" + gradientName + ")");
00663                   }
00664                   else if (FillStyle is RadialGradientBrush radialGradient)
00665                   {
00666                       textElement.SetAttribute("fill", "url(#" + gradientName + ")");
00667                   }
00668
00669                   textElement.SetAttribute("font-size",
    Font.FontSize.ToString(System.Globalization.CultureInfo.InvariantCulture));
00670                   textElement.SetAttribute("font-family", Font.FontFamily.FileName);
```

```
00671
00672                     if (Font.FontFamily.IsBold)
00673                     {
00674                         textElement.SetAttribute("font-weight", "bold");
00675                     }
00676                     else
00677                     {
00678                         textElement.SetAttribute("font-weight", "regular");
00679                     }
00680
00681                     if (Font.FontFamily.IsItalic)
00682                     {
00683                         textElement.SetAttribute("font-style", "italic");
00684                     }
00685                     else
00686                     {
00687                         textElement.SetAttribute("font-style", "normal");
00688                     }
00689
00690                     if (Font.FontFamily.IsOblique)
00691                     {
00692                         textElement.SetAttribute("font-style", "oblique");
00693                     }
00694                 }
00695                 else
00696                 {
00697                     Dictionary<string, string> style = new Dictionary<string, string>();
00698
00699                     style.Add("stroke", "none");
00700
00701                     if (FillStyle is SolidColourBrush solid)
00702                     {
00703                         style.Add("fill", solid.Colour.ToCSSString(false));
00704                         style.Add("fill-opacity",
      solid.A.ToString(System.Globalization.CultureInfo.InvariantCulture));
00705                     }
00706                     else if (FillStyle is LinearGradientBrush linearGradient)
00707                     {
00708                         style.Add("fill", "url(#" + gradientName + ")");
00709                     }
00710                     else if (FillStyle is RadialGradientBrush radialGradient)
00711                     {
00712                         style.Add("fill", "url(#" + gradientName + ")");
00713                     }
00714
00715                     style.Add("font-size",
      Font.FontSize.ToString(System.Globalization.CultureInfo.InvariantCulture) + "px");
00716                     style.Add("font-family", Font.FontFamily.FileName);
00717
00718                     if (Font.FontFamily.IsBold)
00719                     {
00720                         style.Add("font-weight", "bold");
00721                     }
00722                     else
00723                     {
00724                         style.Add("font-weight", "regular");
00725                     }
00726
00727                     if (Font.FontFamily.IsItalic)
00728                     {
00729                         style["font-style"] = "italic";
00730                     }
00731                     else
00732                     {
00733                         style["font-style"] = "normal";
00734                     }
00735
00736                     if (Font.FontFamily.IsOblique)
00737                     {
00738                         style["font-style"] = "oblique";
00739                     }
00740
00741                     string className = GetClass(style);
00742
00743                     textElement.SetAttribute("class", className);
00744                 }
00745
00746                 if (TextOption != SVGContextInterpreter.TextOptions.ConvertIntoPathsUsingGlyphs)
00747                 {
00748                     ProcessText(text, textElement);
00749                 }
00750                 else
00751                 {
00752                     ProcessGlyphs(text, textElement);
00753                 }
00754
00755                 if (!string.IsNullOrEmpty(Tag))
```

```
00756                 {
00757                     textElement.SetAttribute("id", this.TagPrefix + Tag);
00758                 }
00759 
00760                 if (!string.IsNullOrEmpty(this.Tag) && this.linkDestinations.TryGetValue(this.Tag, out
      string destination) && !string.IsNullOrEmpty(destination))
00761                 {
00762                     XmlElement aElement = Document.CreateElement("a", SVGNamespace);
00763                     aElement.SetAttribute("href", destination);
00764                     currentElement.AppendChild(aElement);
00765                     currentElement = aElement;
00766                 }
00767 
00768                 currentElement.AppendChild(textElement);
00769 
00770                 currentElement = currElement;
00771             }
00772             else
00773             {
00774                 PathText(text, x, y);
00775                 Fill(FillRule.NonZeroWinding);
00776             }
00777         }
00778 
00779         public void LineTo(double x, double y)
00780         {
00781             if (currentFigure.PointCount == 0)
00782             {
00783                 currentFigure.StartPoint = new Point(x, y);
00784             }
00785 
00786             currentFigure.CurrentPoint = new Point(x, y);
00787             currentFigure.Data += "L " + x.ToString(System.Globalization.CultureInfo.InvariantCulture)
      + " " + y.ToString(System.Globalization.CultureInfo.InvariantCulture) + " ";
00788             currentFigure.PointCount++;
00789         }
00790 
00791         public void MoveTo(double x, double y)
00792         {
00793             if (currentFigure.PointCount > 0)
00794             {
00795                 currentPath.Figures.Add(currentFigure);
00796             }
00797 
00798             currentFigure = new SVGFigure();
00799             currentFigure.CurrentPoint = new Point(x, y);
00800             currentFigure.StartPoint = new Point(x, y);
00801             currentFigure.Data += "M " + x.ToString(System.Globalization.CultureInfo.InvariantCulture)
      + " " + y.ToString(System.Globalization.CultureInfo.InvariantCulture) + " ";
00802             currentFigure.PointCount = 1;
00803         }
00804 
00805         public void Rectangle(double x0, double y0, double width, double height)
00806         {
00807             MoveTo(x0, y0);
00808             LineTo(x0 + width, y0);
00809             LineTo(x0 + width, y0 + height);
00810             LineTo(x0, y0 + height);
00811             Close();
00812         }
00813 
00814         public void Restore()
00815         {
00816             _transform = states.Pop();
00817             _currClipPath = clipPaths.Pop();
00818             currentPath = new SVGPathObject();
00819             currentFigure = new SVGFigure();
00820         }
00821 
00822         public void Rotate(double angle)
00823         {
00824             _transform = MatrixUtils.Rotate(_transform, angle);
00825             currentPath = new SVGPathObject(); currentPath = new SVGPathObject();
00826             currentFigure = new SVGFigure();
00827         }
00828 
00829         public void Save()
00830         {
00831             states.Push((double[,])_transform.Clone());
00832             clipPaths.Push(_currClipPath);
00833         }
00834 
00835         public void Scale(double scaleX, double scaleY)
00836         {
00837             _transform = MatrixUtils.Scale(_transform, scaleX, scaleY);
00838             currentPath = new SVGPathObject();
00839             currentFigure = new SVGFigure();
```

```
00840         }
00841
00842         public void SetFillStyle((int r, int g, int b, double a) style)
00843         {
00844             FillStyle = Colour.FromRgba(style);
00845         }
00846
00847         public void SetFillStyle(Brush style)
00848         {
00849             FillStyle = style;
00850         }
00851
00852         public void SetLineDash(LineDash dash)
00853         {
00854             _lineDash = dash;
00855         }
00856
00857         public void SetStrokeStyle((int r, int g, int b, double a) style)
00858         {
00859             StrokeStyle = Colour.FromRgba(style);
00860         }
00861
00862         public void SetStrokeStyle(Brush style)
00863         {
00864             StrokeStyle = style;
00865         }
00866
00867         public void Stroke()
00868         {
00869             if (currentFigure.PointCount > 0)
00870             {
00871                 currentPath.Figures.Add(currentFigure);
00872             }
00873
00874             XmlElement currElement = currentElement;
00875
00876             if (!string.IsNullOrEmpty(_currClipPath))
00877             {
00878                 currentElement = Document.CreateElement("g", SVGNamespace);
00879                 currentElement.SetAttribute("clip-path", _currClipPath);
00880                 currElement.AppendChild(currentElement);
00881             }
00882
00883             XmlElement path = Document.CreateElement("path", SVGNamespace);
00884             path.SetAttribute("d", currentPath.Figures.Aggregate("", (a, b) => a + b.Data));
00885
00886             string gradientName = null;
00887
00888             {
00889                 if (StrokeStyle is LinearGradientBrush linearGradient)
00890                 {
00891                     if (!ReuseGradients || !gradients.TryGetValue(linearGradient, out gradientName))
00892                     {
00893                         if (!string.IsNullOrEmpty(Tag))
00894                         {
00895                             gradientName = this.TagPrefix + "gradient" + (gradientCount +
    1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Tag;
00896                         }
00897                         else
00898                         {
00899                             gradientName = this.TagPrefix + "gradient" + (gradientCount +
    1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Guid.NewGuid().ToString("N");
00900                         }
00901
00902                         XmlElement gradientElement = linearGradient.ToLinearGradient(Document,
    gradientName);
00903                         this.Definitions.AppendChild(gradientElement);
00904
00905                         if (ReuseGradients)
00906                         {
00907                             gradients.Add(linearGradient, gradientName);
00908                         }
00909
00910                         gradientCount++;
00911                     }
00912                 }
00913                 else if (StrokeStyle is RadialGradientBrush radialGradient)
00914                 {
00915                     if (!ReuseGradients || !gradients.TryGetValue(radialGradient, out gradientName))
00916                     {
00917                         if (!string.IsNullOrEmpty(Tag))
00918                         {
00919                             gradientName = this.TagPrefix + "gradient" + (gradientCount +
    1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Tag;
00920                         }
00921                         else
00922                         {
```

```
00923                           gradientName = this.TagPrefix + "gradient" + (gradientCount +
1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Guid.NewGuid().ToString("N");
00924                       }
00925
00926                       XmlElement gradientElement = radialGradient.ToRadialGradient(Document,
gradientName);
00927                       this.Definitions.AppendChild(gradientElement);
00928
00929                       if (ReuseGradients)
00930                       {
00931                           gradients.Add(radialGradient, gradientName);
00932                       }
00933
00934                       gradientCount++;
00935                   }
00936               }
00937           }
00938
00939           if (!UseStyles)
00940           {
00941
00942               if (StrokeStyle is SolidColourBrush solid)
00943               {
00944                   path.SetAttribute("stroke", solid.Colour.ToCSSString(false));
00945                   path.SetAttribute("stroke-opacity",
solid.A.ToString(System.Globalization.CultureInfo.InvariantCulture));
00946               }
00947               else if (StrokeStyle is LinearGradientBrush linearGradient)
00948               {
00949                   path.SetAttribute("stroke", "url(#" + gradientName + ")");
00950               }
00951               else if (StrokeStyle is RadialGradientBrush radialGradient)
00952               {
00953                   path.SetAttribute("stroke", "url(#" + gradientName + ")");
00954               }
00955
00956               path.SetAttribute("stroke-width",
LineWidth.ToString(System.Globalization.CultureInfo.InvariantCulture));
00957
00958               switch (LineCap)
00959               {
00960                   case LineCaps.Butt:
00961                       path.SetAttribute("stroke-linecap", "butt");
00962                       break;
00963                   case LineCaps.Round:
00964                       path.SetAttribute("stroke-linecap", "round");
00965                       break;
00966                   case LineCaps.Square:
00967                       path.SetAttribute("stroke-linecap", "square");
00968                       break;
00969               }
00970
00971               switch (LineJoin)
00972               {
00973                   case LineJoins.Bevel:
00974                       path.SetAttribute("stroke-linejoin", "bevel");
00975                       break;
00976                   case LineJoins.Round:
00977                       path.SetAttribute("stroke-linejoin", "round");
00978                       break;
00979                   case LineJoins.Miter:
00980                       path.SetAttribute("stroke-linejoin", "miter");
00981                       break;
00982               }
00983
00984               if (_lineDash.Phase != 0 || (_lineDash.DashArray != null && _lineDash.DashArray.Any(x
=> x != 0)))
00985               {
00986                   string dashArray;
00987
00988                   if (_lineDash.DashArray != null && _lineDash.DashArray.Length > 0)
00989                   {
00990                       StringBuilder dashArraySB = new StringBuilder();
00991                       for (int i = 0; i < _lineDash.DashArray.Length; i++)
00992                       {
00993
dashArraySB.Append(_lineDash.DashArray[i].ToString(System.Globalization.CultureInfo.InvariantCulture));
00994                           dashArraySB.Append("px");
00995                           if (i < _lineDash.DashArray.Length - 1)
00996                           {
00997                               dashArraySB.Append(" ");
00998                           }
00999                       }
01000                       dashArray = dashArraySB.ToString();
01001                   }
01002                   else
01003                   {
```

```
01004                       dashArray = "0px 0px";
01005                   }
01006
01007                   path.SetAttribute("stroke-dasharray", dashArray);
01008                   path.SetAttribute("stroke-dashoffset",
      _lineDash.Phase.ToString(System.Globalization.CultureInfo.InvariantCulture) + "px");
01009               }
01010
01011               path.SetAttribute("fill", "none");
01012               path.SetAttribute("transform", "matrix(" + _transform[0,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + _transform[1,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01013                   "," + _transform[0, 1].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01014                   "," + _transform[0, 2].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
01015
01016           }
01017           else
01018           {
01019               Dictionary<string, string> style = new Dictionary<string, string>();
01020
01021               if (StrokeStyle is SolidColourBrush solid)
01022               {
01023                   style.Add("stroke", solid.Colour.ToCSSString(false));
01024                   style.Add("stroke-opacity",
      solid.A.ToString(System.Globalization.CultureInfo.InvariantCulture));
01025               }
01026               else if (StrokeStyle is LinearGradientBrush linearGradient)
01027               {
01028                   style.Add("stroke", "url(#" + gradientName + ")");
01029               }
01030               else if (StrokeStyle is RadialGradientBrush radialGradient)
01031               {
01032                   style.Add("stroke", "url(#" + gradientName + ")");
01033               }
01034
01035               style.Add("stroke-width",
      LineWidth.ToString(System.Globalization.CultureInfo.InvariantCulture) + "px");
01036
01037               switch (LineCap)
01038               {
01039                   case LineCaps.Butt:
01040                       style.Add("stroke-linecap", "butt");
01041                       break;
01042                   case LineCaps.Round:
01043                       style.Add("stroke-linecap", "round");
01044                       break;
01045                   case LineCaps.Square:
01046                       style.Add("stroke-linecap", "square");
01047                       break;
01048               }
01049
01050               switch (LineJoin)
01051               {
01052                   case LineJoins.Bevel:
01053                       style.Add("stroke-linejoin", "bevel");
01054                       break;
01055                   case LineJoins.Round:
01056                       style.Add("stroke-linejoin", "round");
01057                       break;
01058                   case LineJoins.Miter:
01059                       style.Add("stroke-linejoin", "miter");
01060                       break;
01061               }
01062
01063               if (_lineDash.Phase != 0 || (_lineDash.DashArray != null && _lineDash.DashArray.Any(x
      => x != 0)))
01064               {
01065                   string dashArray;
01066
01067                   if (_lineDash.DashArray != null && _lineDash.DashArray.Length > 0)
01068                   {
01069                       StringBuilder dashArraySB = new StringBuilder();
01070                       for (int i = 0; i < _lineDash.DashArray.Length; i++)
01071                       {
01072
      dashArraySB.Append(_lineDash.DashArray[i].ToString(System.Globalization.CultureInfo.InvariantCulture));
01073                           dashArraySB.Append("px");
01074                           if (i < _lineDash.DashArray.Length - 1)
01075                           {
01076                               dashArraySB.Append(" ");
01077                           }
01078                       }
01079                       dashArray = dashArraySB.ToString();
01080                   }
01081                   else
```

```
01082                     {
01083                         dashArray = "0px 0px";
01084                     }
01085 
01086                     style.Add("stroke-dasharray", dashArray);
01087                     style.Add("stroke-dashoffset",
      _lineDash.Phase.ToString(System.Globalization.CultureInfo.InvariantCulture) + "px");
01088                 }
01089 
01090                 style.Add("fill", "none");
01091                 style.Add("transform", "matrix(" + _transform[0,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + _transform[1,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01092                     "," + _transform[0, 1].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01093                     "," + _transform[0, 2].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
01094 
01095                 string className = GetClass(style);
01096                 path.SetAttribute("class", className);
01097 
01098             }
01099 
01100             if (!string.IsNullOrEmpty(Tag))
01101             {
01102                 path.SetAttribute("id", this.TagPrefix + Tag);
01103             }
01104 
01105             if (!string.IsNullOrEmpty(this.Tag) && this.linkDestinations.TryGetValue(this.Tag, out
      string destination) && !string.IsNullOrEmpty(destination))
01106             {
01107                 XmlElement aElement = Document.CreateElement("a", SVGNamespace);
01108                 aElement.SetAttribute("href", destination);
01109                 currentElement.AppendChild(aElement);
01110                 currentElement = aElement;
01111             }
01112 
01113             currentElement.AppendChild(path);
01114 
01115             currentElement = currElement;
01116 
01117             currentPath = new SVGPathObject();
01118             currentFigure = new SVGFigure();
01119         }
01120 
01121         public void StrokeText(string text, double x, double y)
01122         {
01123             if ((!TextToPaths && TextOption !=
      SVGContextInterpreter.TextOptions.ConvertIntoPathsUsingGlyphs) || (TextOption ==
      SVGContextInterpreter.TextOptions.ConvertIntoPathsUsingGlyphs && StrokeStyle is SolidColourBrush))
01124             {
01125                 if (!UsedFontFamilies.ContainsKey(Font.FontFamily.FileName))
01126                 {
01127                     UsedFontFamilies.Add(Font.FontFamily.FileName, Font.FontFamily);
01128                     UsedChars.Add(Font.FontFamily.FileName, new HashSet<char>());
01129                 }
01130 
01131                 UsedChars[Font.FontFamily.FileName].UnionWith(text);
01132 
01133                 Font.DetailedFontMetrics metrics = Font.MeasureTextAdvanced(text);
01134 
01135                 double[,] currTransform = null;
01136                 double[,] deltaTransform = MatrixUtils.Identity;
01137 
01138                 switch (TextBaseline)
01139                 {
01140                     case TextBaselines.Baseline:
01141                         currTransform = MatrixUtils.Translate(_transform, x - metrics.LeftSideBearing,
      y);
01142                         deltaTransform = MatrixUtils.Translate(deltaTransform, x -
      metrics.LeftSideBearing, y);
01143                         break;
01144                     case TextBaselines.Top:
01145                         currTransform = MatrixUtils.Translate(_transform, x - metrics.LeftSideBearing,
      y + metrics.Top);
01146                         deltaTransform = MatrixUtils.Translate(deltaTransform, x -
      metrics.LeftSideBearing, y + metrics.Top);
01147                         break;
01148                     case TextBaselines.Bottom:
01149                         currTransform = MatrixUtils.Translate(_transform, x - metrics.LeftSideBearing,
      y + metrics.Bottom);
01150                         deltaTransform = MatrixUtils.Translate(deltaTransform, x -
      metrics.LeftSideBearing, y + metrics.Bottom);
01151                         break;
01152                     case TextBaselines.Middle:
01153                         currTransform = MatrixUtils.Translate(_transform, x - metrics.LeftSideBearing,
      y + (metrics.Top + metrics.Bottom) * 0.5);
```

```
01154                           deltaTransform = MatrixUtils.Translate(deltaTransform, x -
      metrics.LeftSideBearing, y + (metrics.Top + metrics.Bottom) * 0.5);
01155                           break;
01156                       default:
01157                           currTransform = MatrixUtils.Translate(_transform, x - metrics.LeftSideBearing,
      y);
01158                           deltaTransform = MatrixUtils.Translate(deltaTransform, x -
      metrics.LeftSideBearing, y);
01159                           break;
01160                   }
01161
01162                   XmlElement currElement = currentElement;
01163
01164                   if (!string.IsNullOrEmpty(_currClipPath))
01165                   {
01166                       currentElement = Document.CreateElement("g", SVGNamespace);
01167                       currentElement.SetAttribute("clip-path", _currClipPath);
01168                       currElement.AppendChild(currentElement);
01169                   }
01170
01171                   XmlElement textElement;
01172
01173                   if (TextOption != SVGContextInterpreter.TextOptions.ConvertIntoPathsUsingGlyphs)
01174                   {
01175                       textElement = Document.CreateElement("text", SVGNamespace);
01176                   }
01177                   else
01178                   {
01179                       textElement = Document.CreateElement("g", SVGNamespace);
01180                   }
01181
01182
01183                   string gradientName = null;
01184
01185                   {
01186                       if (StrokeStyle is LinearGradientBrush linearGradient)
01187                       {
01188                           if (!string.IsNullOrEmpty(Tag))
01189                           {
01190                               gradientName = this.TagPrefix + "textGradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Tag;
01191                           }
01192                           else
01193                           {
01194                               gradientName = this.TagPrefix + "textGradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Guid.NewGuid().ToString("N");
01195                           }
01196
01197                           XmlElement gradientElement = linearGradient.ToLinearGradient(Document,
      gradientName);
01198
01199                           deltaTransform = MatrixUtils.Invert(deltaTransform);
01200
01201                           gradientElement.SetAttribute("gradientTransform", "matrix(" +
      deltaTransform[0, 0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," +
      deltaTransform[1, 0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01202                               "," + deltaTransform[0,
      1].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + deltaTransform[1,
      1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01203                               "," + deltaTransform[0,
      2].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + deltaTransform[1,
      2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
01204
01205                           this.Definitions.AppendChild(gradientElement);
01206
01207                           gradientCount++;
01208                       }
01209                       else if (StrokeStyle is RadialGradientBrush radialGradient)
01210                       {
01211                           if (!string.IsNullOrEmpty(Tag))
01212                           {
01213                               gradientName = this.TagPrefix + "textGradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Tag;
01214                           }
01215                           else
01216                           {
01217                               gradientName = this.TagPrefix + "textGradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Guid.NewGuid().ToString("N");
01218                           }
01219
01220                           XmlElement gradientElement = radialGradient.ToRadialGradient(Document,
      gradientName);
01221
01222                           deltaTransform = MatrixUtils.Invert(deltaTransform);
01223
01224                           gradientElement.SetAttribute("gradientTransform", "matrix(" +
      deltaTransform[0, 0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," +
```

```
      deltaTransform[1, 0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01225                         "," + deltaTransform[0,
      1].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + deltaTransform[1,
      1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01226                         "," + deltaTransform[0,
      2].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + deltaTransform[1,
      2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
01227
01228                         this.Definitions.AppendChild(gradientElement);
01229
01230                         gradientCount++;
01231                     }
01232                 }
01233
01234                 if (TextOption != SVGContextInterpreter.TextOptions.ConvertIntoPathsUsingGlyphs)
01235                 {
01236                     textElement.SetAttribute("x", "0");
01237                     textElement.SetAttribute("y", "0");
01238                 }
01239
01240                 textElement.SetAttribute("transform", "matrix(" + currTransform[0,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + currTransform[1,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01241                         "," + currTransform[0,
      1].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + currTransform[1,
      1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01242                         "," + currTransform[0,
      2].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + currTransform[1,
      2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
01243
01244                 if (!UseStyles)
01245                 {
01246                     if (StrokeStyle is SolidColourBrush solid)
01247                     {
01248                         textElement.SetAttribute("stroke", solid.Colour.ToCSSString(false));
01249                         textElement.SetAttribute("stroke-opacity",
      solid.A.ToString(System.Globalization.CultureInfo.InvariantCulture));
01250                     }
01251                     else if (StrokeStyle is LinearGradientBrush linearGradient)
01252                     {
01253                         textElement.SetAttribute("stroke", "url(#" + gradientName + ")");
01254                     }
01255                     else if (StrokeStyle is RadialGradientBrush radialGradient)
01256                     {
01257                         textElement.SetAttribute("stroke", "url(#" + gradientName + ")");
01258                     }
01259
01260                     if (TextOption != SVGContextInterpreter.TextOptions.ConvertIntoPathsUsingGlyphs)
01261                     {
01262                         textElement.SetAttribute("stroke-width",
      LineWidth.ToString(System.Globalization.CultureInfo.InvariantCulture));
01263                     }
01264                     else
01265                     {
01266                         textElement.SetAttribute("stroke-width", (LineWidth *
      Font.FontFamily.TrueTypeFile.GetUnitsPerEm() /
      Font.FontSize).ToString(System.Globalization.CultureInfo.InvariantCulture));
01267                     }
01268
01269
01270                     switch (LineCap)
01271                     {
01272                         case LineCaps.Butt:
01273                             textElement.SetAttribute("stroke-linecap", "butt");
01274                             break;
01275                         case LineCaps.Round:
01276                             textElement.SetAttribute("stroke-linecap", "round");
01277                             break;
01278                         case LineCaps.Square:
01279                             textElement.SetAttribute("stroke-linecap", "square");
01280                             break;
01281                     }
01282
01283                     switch (LineJoin)
01284                     {
01285                         case LineJoins.Bevel:
01286                             textElement.SetAttribute("stroke-linejoin", "bevel");
01287                             break;
01288                         case LineJoins.Round:
01289                             textElement.SetAttribute("stroke-linejoin", "round");
01290                             break;
01291                         case LineJoins.Miter:
01292                             textElement.SetAttribute("stroke-linejoin", "miter");
01293                             break;
01294                     }
01295
01296                     if (_lineDash.Phase != 0 || (_lineDash.DashArray != null &&
```

```
    _lineDash.DashArray.Any(v => v != 0)))
01297                     {
01298                         string dashArray;
01299 
01300                         if (_lineDash.DashArray != null && _lineDash.DashArray.Length > 0)
01301                         {
01302                             StringBuilder dashArraySB = new StringBuilder();
01303                             for (int i = 0; i < _lineDash.DashArray.Length; i++)
01304                             {
01305 
    dashArraySB.Append(_lineDash.DashArray[i].ToString(System.Globalization.CultureInfo.InvariantCulture));
01306                                 dashArraySB.Append("px");
01307                                 if (i < _lineDash.DashArray.Length - 1)
01308                                 {
01309                                     dashArraySB.Append(" ");
01310                                 }
01311                             }
01312                             dashArray = dashArraySB.ToString();
01313                         }
01314                         else
01315                         {
01316                             dashArray = "0px 0px";
01317                         }
01318 
01319                         textElement.SetAttribute("stroke-dasharray", dashArray);
01320                         textElement.SetAttribute("stroke-dashoffset",
    _lineDash.Phase.ToString(System.Globalization.CultureInfo.InvariantCulture) + "px");
01321                     }
01322                     textElement.SetAttribute("fill", "none");
01323 
01324                     textElement.SetAttribute("font-size",
    Font.FontSize.ToString(System.Globalization.CultureInfo.InvariantCulture));
01325                     textElement.SetAttribute("font-family", Font.FontFamily.FileName);
01326 
01327                     if (Font.FontFamily.IsBold)
01328                     {
01329                         textElement.SetAttribute("font-weight", "bold");
01330                     }
01331                     else
01332                     {
01333                         textElement.SetAttribute("font-weight", "regular");
01334                     }
01335 
01336                     if (Font.FontFamily.IsItalic)
01337                     {
01338                         textElement.SetAttribute("font-style", "italic");
01339                     }
01340                     else
01341                     {
01342                         textElement.SetAttribute("font-style", "normal");
01343                     }
01344 
01345                     if (Font.FontFamily.IsOblique)
01346                     {
01347                         textElement.SetAttribute("font-style", "oblique");
01348                     }
01349                 }
01350                 else
01351                 {
01352                     Dictionary<string, string> style = new Dictionary<string, string>();
01353 
01354                     if (StrokeStyle is SolidColourBrush solid)
01355                     {
01356                         style.Add("stroke", solid.Colour.ToCSSString(false));
01357                         style.Add("stroke-opacity",
    solid.A.ToString(System.Globalization.CultureInfo.InvariantCulture));
01358                     }
01359                     else if (StrokeStyle is LinearGradientBrush linearGradient)
01360                     {
01361                         style.Add("stroke", "url(#" + gradientName + ")");
01362                     }
01363                     else if (StrokeStyle is RadialGradientBrush radialGradient)
01364                     {
01365                         style.Add("stroke", "url(#" + gradientName + ")");
01366                     }
01367 
01368                     style.Add("stroke-width",
    LineWidth.ToString(System.Globalization.CultureInfo.InvariantCulture) + "px");
01369 
01370                     switch (LineCap)
01371                     {
01372                         case LineCaps.Butt:
01373                             style.Add("stroke-linecap", "butt");
01374                             break;
01375                         case LineCaps.Round:
01376                             style.Add("stroke-linecap", "round");
01377                             break;
```

```
01378                         case LineCaps.Square:
01379                             style.Add("stroke-linecap", "square");
01380                             break;
01381                     }
01382 
01383                     switch (LineJoin)
01384                     {
01385                         case LineJoins.Bevel:
01386                             style.Add("stroke-linejoin", "bevel");
01387                             break;
01388                         case LineJoins.Round:
01389                             style.Add("stroke-linejoin", "round");
01390                             break;
01391                         case LineJoins.Miter:
01392                             style.Add("stroke-linejoin", "miter");
01393                             break;
01394                     }
01395 
01396                     if (_lineDash.Phase != 0 || (_lineDash.DashArray != null &&
    _lineDash.DashArray.Any(v => v != 0)))
01397                     {
01398                         string dashArray;
01399 
01400                         if (_lineDash.DashArray != null && _lineDash.DashArray.Length > 0)
01401                         {
01402                             StringBuilder dashArraySB = new StringBuilder();
01403                             for (int i = 0; i < _lineDash.DashArray.Length; i++)
01404                             {
01405 
    dashArraySB.Append(_lineDash.DashArray[i].ToString(System.Globalization.CultureInfo.InvariantCulture));
01406                                 dashArraySB.Append("px");
01407                                 if (i < _lineDash.DashArray.Length - 1)
01408                                 {
01409                                     dashArraySB.Append(" ");
01410                                 }
01411                             }
01412                             dashArray = dashArraySB.ToString();
01413                         }
01414                         else
01415                         {
01416                             dashArray = "0px 0px";
01417                         }
01418 
01419                         style.Add("stroke-dasharray", dashArray);
01420                         style.Add("stroke-dashoffset",
    _lineDash.Phase.ToString(System.Globalization.CultureInfo.InvariantCulture) + "px");
01421                     }
01422                     style.Add("fill", "none");
01423 
01424                     style.Add("font-size",
    Font.FontSize.ToString(System.Globalization.CultureInfo.InvariantCulture) + "px");
01425                     style.Add("font-family", Font.FontFamily.FileName);
01426 
01427                     if (Font.FontFamily.IsBold)
01428                     {
01429                         style.Add("font-weight", "bold");
01430                     }
01431                     else
01432                     {
01433                         style.Add("font-weight", "regular");
01434                     }
01435 
01436                     if (Font.FontFamily.IsItalic)
01437                     {
01438                         style["font-style"] = "italic";
01439                     }
01440                     else
01441                     {
01442                         style["font-style"] = "normal";
01443                     }
01444 
01445                     if (Font.FontFamily.IsOblique)
01446                     {
01447                         style["font-style"] = "oblique";
01448                     }
01449 
01450                     string className = GetClass(style);
01451 
01452                     textElement.SetAttribute("class", className);
01453                 }
01454 
01455                 if (TextOption != SVGContextInterpreter.TextOptions.ConvertIntoPathsUsingGlyphs)
01456                 {
01457                     ProcessText(text, textElement);
01458                 }
01459                 else
01460                 {
```

```
01461                   ProcessGlyphs(text, textElement);
01462               }
01463
01464               if (!string.IsNullOrEmpty(Tag))
01465               {
01466                   textElement.SetAttribute("id", this.TagPrefix + Tag);
01467               }
01468
01469               if (!string.IsNullOrEmpty(this.Tag) && this.linkDestinations.TryGetValue(this.Tag, out
      string destination) && !string.IsNullOrEmpty(destination))
01470               {
01471                   XmlElement aElement = Document.CreateElement("a", SVGNamespace);
01472                   aElement.SetAttribute("href", destination);
01473                   currentElement.AppendChild(aElement);
01474                   currentElement = aElement;
01475               }
01476
01477               currentElement.AppendChild(textElement);
01478
01479               currentElement = currElement;
01480           }
01481           else
01482           {
01483               PathText(text, x, y);
01484               Stroke();
01485           }
01486       }
01487
01488       public void Transform(double a, double b, double c, double d, double e, double f)
01489       {
01490           double[,] transfMatrix = new double[3, 3] { { a, c, e }, { b, d, f }, { 0, 0, 1 } };
01491           _transform = MatrixUtils.Multiply(_transform, transfMatrix);
01492
01493           currentPath = new SVGPathObject();
01494           currentFigure = new SVGFigure();
01495       }
01496
01497       public void Translate(double x, double y)
01498       {
01499           _transform = MatrixUtils.Translate(_transform, x, y);
01500
01501           currentPath = new SVGPathObject();
01502           currentFigure = new SVGFigure();
01503       }
01504
01505       public void SetClippingPath()
01506       {
01507           if (currentFigure.PointCount > 0)
01508           {
01509               currentPath.Figures.Add(currentFigure);
01510           }
01511
01512           XmlElement clipPath = Document.CreateElement("clipPath", SVGNamespace);
01513           string id = !string.IsNullOrEmpty(Tag) ?  (this.TagPrefix + Tag) :
      Guid.NewGuid().ToString("N");
01514           clipPath.SetAttribute("id", id);
01515
01516           if (!string.IsNullOrEmpty(_currClipPath))
01517           {
01518               clipPath.SetAttribute("clip-path", _currClipPath);
01519           }
01520
01521           XmlElement path = Document.CreateElement("path", SVGNamespace);
01522           path.SetAttribute("id", id + "_clipPath");
01523           path.SetAttribute("d", currentPath.Figures.Aggregate("", (a, b) => a + b.Data));
01524
01525           if (StrokeStyle is SolidColourBrush solid)
01526           {
01527               path.SetAttribute("stroke", solid.Colour.ToCSSString(false));
01528               path.SetAttribute("stroke-opacity",
      solid.A.ToString(System.Globalization.CultureInfo.InvariantCulture));
01529           }
01530           else if (StrokeStyle is LinearGradientBrush linearGradient)
01531           {
01532               string gradientName;
01533
01534               if (!ReuseGradients || !gradients.TryGetValue(linearGradient, out gradientName))
01535               {
01536                   if (!string.IsNullOrEmpty(Tag))
01537                   {
01538                       gradientName = this.TagPrefix + "gradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Tag;
01539                   }
01540                   else
01541                   {
01542                       gradientName = this.TagPrefix + "gradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Guid.NewGuid().ToString("N");
```

```
01543                     }
01544
01545                     XmlElement gradientElement = linearGradient.ToLinearGradient(Document,
      gradientName);
01546                     this.Definitions.AppendChild(gradientElement);
01547
01548                     if (ReuseGradients)
01549                     {
01550                         gradients.Add(linearGradient, gradientName);
01551                     }
01552
01553                     gradientCount++;
01554                 }
01555
01556                 path.SetAttribute("stroke", "url(#" + gradientName + ")");
01557             }
01558             else if (StrokeStyle is RadialGradientBrush radialGradient)
01559             {
01560                 string gradientName;
01561
01562                 if (!ReuseGradients || !gradients.TryGetValue(radialGradient, out gradientName))
01563                 {
01564                     if (!string.IsNullOrEmpty(Tag))
01565                     {
01566                         gradientName = this.TagPrefix + "gradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Tag;
01567                     }
01568                     else
01569                     {
01570                         gradientName = this.TagPrefix + "gradient" + (gradientCount +
      1).ToString(System.Globalization.CultureInfo.InvariantCulture) + "_" + Guid.NewGuid().ToString("N");
01571                     }
01572
01573                     XmlElement gradientElement = radialGradient.ToRadialGradient(Document,
      gradientName);
01574                     this.Definitions.AppendChild(gradientElement);
01575
01576                     if (ReuseGradients)
01577                     {
01578                         gradients.Add(radialGradient, gradientName);
01579                     }
01580
01581                     gradientCount++;
01582                 }
01583
01584                 path.SetAttribute("stroke", "url(#" + gradientName + ")");
01585             }
01586
01587             path.SetAttribute("stroke-width",
      LineWidth.ToString(System.Globalization.CultureInfo.InvariantCulture));
01588
01589             switch (LineCap)
01590             {
01591                 case LineCaps.Butt:
01592                     path.SetAttribute("stroke-linecap", "butt");
01593                     break;
01594                 case LineCaps.Round:
01595                     path.SetAttribute("stroke-linecap", "round");
01596                     break;
01597                 case LineCaps.Square:
01598                     path.SetAttribute("stroke-linecap", "square");
01599                     break;
01600             }
01601
01602             switch (LineJoin)
01603             {
01604                 case LineJoins.Bevel:
01605                     path.SetAttribute("stroke-linejoin", "bevel");
01606                     break;
01607                 case LineJoins.Round:
01608                     path.SetAttribute("stroke-linejoin", "round");
01609                     break;
01610                 case LineJoins.Miter:
01611                     path.SetAttribute("stroke-linejoin", "miter");
01612                     break;
01613             }
01614
01615             if (_lineDash.Phase != 0 || (_lineDash.DashArray != null && _lineDash.DashArray.Any(x => x
      != 0)))
01616             {
01617                 string dashArray;
01618
01619                 if (_lineDash.DashArray != null && _lineDash.DashArray.Length > 0)
01620                 {
01621                     StringBuilder dashArraySB = new StringBuilder();
01622                     for (int i = 0; i < _lineDash.DashArray.Length; i++)
01623                     {
```

```
01624
      dashArraySB.Append(_lineDash.DashArray[i].ToString(System.Globalization.CultureInfo.InvariantCulture));
01625                         dashArraySB.Append("px");
01626                         if (i < _lineDash.DashArray.Length - 1)
01627                         {
01628                             dashArraySB.Append(" ");
01629                         }
01630                     }
01631                     dashArray = dashArraySB.ToString();
01632                 }
01633                 else
01634                 {
01635                     dashArray = "0px 0px";
01636                 }
01637
01638                 path.SetAttribute("stroke-dasharray", dashArray);
01639                 path.SetAttribute("stroke-dashoffset",
      _lineDash.Phase.ToString(System.Globalization.CultureInfo.InvariantCulture) + "px");
01640             }
01641
01642             path.SetAttribute("fill", "none");
01643             path.SetAttribute("transform", "matrix(" + _transform[0,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + _transform[1,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01644                 "," + _transform[0, 1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
      "," + _transform[1, 1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01645                 "," + _transform[0, 2].ToString(System.Globalization.CultureInfo.InvariantCulture) +
      "," + _transform[1, 2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
01646
01647             clipPath.AppendChild(path);
01648
01649             currentElement.AppendChild(clipPath);
01650
01651             _currClipPath = "url(#" + id + ")";
01652
01653             currentPath = new SVGPathObject();
01654             currentFigure = new SVGFigure();
01655         }
01656
01657         public void DrawRasterImage(int sourceX, int sourceY, int sourceWidth, int sourceHeight,
      double destinationX, double destinationY, double destinationWidth, double destinationHeight,
      RasterImage image)
01658         {
01659             lock (lockObject)
01660             {
01661                 UsedRasterImages[image.Id] = image;
01662             }
01663
01664             Save();
01665
01666             MoveTo(destinationX, destinationY);
01667             LineTo(destinationX + destinationWidth, destinationY);
01668             LineTo(destinationX + destinationWidth, destinationY + destinationHeight);
01669             LineTo(destinationX, destinationY + destinationHeight);
01670             Close();
01671             SetClippingPath();
01672
01673             double sourceRectX = (double)sourceX / image.Width;
01674             double sourceRectY = (double)sourceY / image.Height;
01675             double sourceRectWidth = (double)sourceWidth / image.Width;
01676             double sourceRectHeight = (double)sourceHeight / image.Height;
01677
01678             double scaleX = destinationWidth / sourceRectWidth;
01679             double scaleY = destinationHeight / sourceRectHeight;
01680
01681             double translationX = destinationX / scaleX - sourceRectX;
01682             double translationY = destinationY / scaleY - sourceRectY;
01683
01684             Scale(scaleX, scaleY);
01685             Translate(translationX, translationY);
01686
01687             XmlElement currElement = currentElement;
01688
01689             if (!string.IsNullOrEmpty(_currClipPath))
01690             {
01691                 currentElement = Document.CreateElement("g", SVGNamespace);
01692                 currentElement.SetAttribute("clip-path", _currClipPath);
01693                 currElement.AppendChild(currentElement);
01694             }
01695
01696             XmlElement img = Document.CreateElement("use", SVGNamespace);
01697
01698             img.SetAttribute("href", "#" + image.Id);
01699
01700             img.SetAttribute("transform", "matrix(" + _transform[0,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + _transform[1,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
```

```
01701                 "," + _transform[0, 1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
      "," + _transform[1, 1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01702                 "," + _transform[0, 2].ToString(System.Globalization.CultureInfo.InvariantCulture) +
      "," + _transform[1, 2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
01703
01704             if (!string.IsNullOrEmpty(Tag))
01705             {
01706                 img.SetAttribute("id", this.TagPrefix + Tag);
01707             }
01708
01709             if (!string.IsNullOrEmpty(this.Tag) && this.linkDestinations.TryGetValue(this.Tag, out
      string destination) && !string.IsNullOrEmpty(destination))
01710             {
01711                 XmlElement aElement = Document.CreateElement("a", SVGNamespace);
01712                 aElement.SetAttribute("href", destination);
01713                 currentElement.AppendChild(aElement);
01714                 currentElement = aElement;
01715             }
01716
01717             currentElement.AppendChild(img);
01718
01719             currentElement = currElement;
01720
01721             Restore();
01722         }
01723
01724         private void ProcessText(string text, XmlNode parent)
01725         {
01726             if (Font.EnableKerning && this.TextOption ==
      SVGContextInterpreter.TextOptions.SubsetFonts)
01727             {
01728                 List<(string, Point)> tSpans = new List<(string, Point)>();
01729
01730                 StringBuilder currentRun = new StringBuilder();
01731                 Point currentKerning = new Point();
01732
01733                 Point currentGlyphPlacementDelta = new Point();
01734                 Point currentGlyphAdvanceDelta = new Point();
01735                 Point nextGlyphPlacementDelta = new Point();
01736                 Point nextGlyphAdvanceDelta = new Point();
01737
01738                 for (int i = 0; i < text.Length; i++)
01739                 {
01740                     if (i < text.Length - 1)
01741                     {
01742                         currentGlyphPlacementDelta = nextGlyphPlacementDelta;
01743                         currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
01744                         nextGlyphAdvanceDelta = new Point();
01745                         nextGlyphPlacementDelta = new Point();
01746
01747                         TrueTypeFile.PairKerning kerning =
      Font.FontFamily.TrueTypeFile.Get1000EmKerning(text[i], text[i + 1]);
01748
01749                         if (kerning != null)
01750                         {
01751                             currentGlyphPlacementDelta = new Point(currentGlyphPlacementDelta.X +
      kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y + kerning.Glyph1Placement.Y);
01752                             currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
      kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
01753
01754                             nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
      kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
01755                             nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
      kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
01756                         }
01757                     }
01758
01759                     if (currentGlyphPlacementDelta.X != 0 || currentGlyphPlacementDelta.Y != 0 ||
      currentGlyphAdvanceDelta.X != 0 || currentGlyphAdvanceDelta.Y != 0)
01760                     {
01761                         if (currentRun.Length > 0)
01762                         {
01763                             tSpans.Add((currentRun.ToString(), currentKerning));
01764
01765                             tSpans.Add((text[i].ToString(), new Point(currentGlyphPlacementDelta.X *
      Font.FontSize / 1000, currentGlyphPlacementDelta.Y * Font.FontSize / 1000)));
01766
01767                             currentRun.Clear();
01768                             currentKerning = new Point((currentGlyphAdvanceDelta.X -
      currentGlyphPlacementDelta.X) * Font.FontSize / 1000, (currentGlyphAdvanceDelta.Y -
      currentGlyphPlacementDelta.Y) * Font.FontSize / 1000);
01769                         }
01770                         else
01771                         {
01772                             tSpans.Add((text[i].ToString(), new Point(currentGlyphPlacementDelta.X *
      Font.FontSize / 1000 + currentKerning.X, currentGlyphPlacementDelta.Y * Font.FontSize / 1000 +
      currentKerning.Y)));
```

```
01773
01774                           currentRun.Clear();
01775                           currentKerning = new Point((currentGlyphAdvanceDelta.X -
      currentGlyphPlacementDelta.X) * Font.FontSize / 1000, (currentGlyphAdvanceDelta.Y -
      currentGlyphPlacementDelta.Y) * Font.FontSize / 1000);
01776                       }
01777                   }
01778                   else
01779                   {
01780                       currentRun.Append(text[i]);
01781                   }
01782               }
01783
01784               if (currentRun.Length > 0)
01785               {
01786                   tSpans.Add((currentRun.ToString(), currentKerning));
01787               }
01788
01789               for (int i = 0; i < tSpans.Count; i++)
01790               {
01791                   XmlElement tspanElement = Document.CreateElement("tspan", SVGNamespace);
01792                   tspanElement.InnerText = tSpans[i].Item1.Replace(" ", "\u00A0");
01793
01794                   if (tSpans[i].Item2.X != 0)
01795                   {
01796                       tspanElement.SetAttribute("dx",
      tSpans[i].Item2.X.ToString(System.Globalization.CultureInfo.InvariantCulture));
01797                   }
01798
01799                   if (tSpans[i].Item2.Y != 0)
01800                   {
01801                       tspanElement.SetAttribute("dy",
      tSpans[i].Item2.Y.ToString(System.Globalization.CultureInfo.InvariantCulture));
01802                   }
01803
01804                   parent.AppendChild(tspanElement);
01805               }
01806
01807           }
01808           else
01809           {
01810               parent.InnerText = text.Replace(" ", "\u00A0");
01811           }
01812       }
01813
01814       private void ProcessGlyphs(string text, XmlNode parent)
01815       {
01816           List<(string, Point)> tSpans = new List<(string, Point)>();
01817
01818           Point currentGlyphPlacementDelta = new Point();
01819           Point currentGlyphAdvanceDelta = new Point();
01820           Point nextGlyphPlacementDelta = new Point();
01821           Point nextGlyphAdvanceDelta = new Point();
01822
01823           double currX = 0;
01824           double currY = 0;
01825
01826           for (int i = 0; i < text.Length; i++)
01827           {
01828               if (i < text.Length - 1)
01829               {
01830                   currentGlyphPlacementDelta = nextGlyphPlacementDelta;
01831                   currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
01832                   nextGlyphAdvanceDelta = new Point();
01833                   nextGlyphPlacementDelta = new Point();
01834
01835                   TrueTypeFile.PairKerning kerning =
      Font.FontFamily.TrueTypeFile.Get1000EmKerning(text[i], text[i + 1]);
01836
01837                   if (kerning != null)
01838                   {
01839                       currentGlyphPlacementDelta = new Point(currentGlyphPlacementDelta.X +
      kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y + kerning.Glyph1Placement.Y);
01840                       currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
      kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
01841
01842                       nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
      kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
01843                       nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
      kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
01844                   }
01845               }
01846
01847               tSpans.Add((text[i].ToString(), new Point(currX, currY)));
01848
01849               double advanceWidth = Font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(text[i]) *
      Font.FontSize / 1000;
```

```
01850                 currX += advanceWidth + currentGlyphPlacementDelta.X * Font.FontSize / 1000 +
      (currentGlyphAdvanceDelta.X - currentGlyphPlacementDelta.X) * Font.FontSize / 1000;
01851                 currY += currentGlyphPlacementDelta.Y * Font.FontSize / 1000 +
      (currentGlyphAdvanceDelta.Y - currentGlyphPlacementDelta.Y) * Font.FontSize / 1000;
01852             }
01853
01854             double scale = Font.FontSize / Font.FontFamily.TrueTypeFile.GetUnitsPerEm();
01855
01856             string scaleText = scale.ToString(System.Globalization.CultureInfo.InvariantCulture);
01857
01858             ((XmlElement)parent).SetAttribute("transform",
      ((XmlElement)parent).GetAttribute("transform") + ", scale(" + scaleText + ")");
01859
01860             for (int i = 0; i < tSpans.Count; i++)
01861             {
01862                 XmlElement useElement = Document.CreateElement("use", SVGNamespace);
01863                 useElement.SetAttribute("href", "#" + Font.FontFamily.FileName + "-" +
      tSpans[i].Item1);
01864
01865                 useElement.SetAttribute("transform", "translate(" + (tSpans[i].Item2.X /
      scale).ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + (tSpans[i].Item2.Y /
      scale).ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
01866                 parent.AppendChild(useElement);
01867             }
01868         }
01869
01870         public void DrawFilteredGraphics(Graphics graphics, IFilter filter)
01871         {
01872             if (FilterOption.Operation ==
      SVGContextInterpreter.FilterOption.FilterOperations.IgnoreAll)
01873             {
01874                 graphics.CopyToIGraphicsContext(this);
01875             }
01876             else if (FilterOption.Operation ==
      SVGContextInterpreter.FilterOption.FilterOperations.SkipAll)
01877             {
01878
01879             }
01880             else
01881             {
01882                 bool rasterisationNeeded = false;
01883                 bool justDraw = false;
01884
01885                 if (FilterOption.Operation ==
      SVGContextInterpreter.FilterOption.FilterOperations.RasteriseAll)
01886                 {
01887                     rasterisationNeeded = true;
01888                     justDraw = false;
01889                 }
01890                 else
01891                 {
01892                     if (FilterOption.Operation ==
      SVGContextInterpreter.FilterOption.FilterOperations.RasteriseIfNecessary)
01893                     {
01894                         justDraw = false;
01895                         rasterisationNeeded = true;
01896                     }
01897                     else if (FilterOption.Operation ==
      SVGContextInterpreter.FilterOption.FilterOperations.NeverRasteriseAndIgnore)
01898                     {
01899                         rasterisationNeeded = false;
01900                         justDraw = true;
01901                     }
01902                     else if (FilterOption.Operation ==
      SVGContextInterpreter.FilterOption.FilterOperations.NeverRasteriseAndSkip)
01903                     {
01904                         rasterisationNeeded = false;
01905                         justDraw = false;
01906                     }
01907
01908                     if (filter is MaskFilter mask)
01909                     {
01910                         rasterisationNeeded = false;
01911                         justDraw = false;
01912
01913                         XmlElement currElement = currentElement;
01914
01915                         if (!string.IsNullOrEmpty(_currClipPath))
01916                         {
01917                             currentElement = Document.CreateElement("g", SVGNamespace);
01918                             currentElement.SetAttribute("clip-path", _currClipPath);
01919                             currElement.AppendChild(currentElement);
01920                         }
01921
01922                         XmlElement currentElement2 = currentElement;
01923
01924                         string filterGuid = !string.IsNullOrEmpty(Tag) ?  (this.TagPrefix + Tag +
```

```
      "@filter") :  Guid.NewGuid().ToString("N");
01925
01926                         Rectangle bounds = graphics.GetBounds();
01927
01928                         Point p1 = new Point(bounds.Location.X, bounds.Location.Y);
01929                         Point p2 = new Point(bounds.Location.X + bounds.Size.Width,
      bounds.Location.Y);
01930                         Point p3 = new Point(bounds.Location.X + bounds.Size.Width, bounds.Location.Y
      + bounds.Size.Height);
01931                         Point p4 = new Point(bounds.Location.X, bounds.Location.Y +
      bounds.Size.Height);
01932
01933                         /*p1 = MatrixUtils.Multiply(_transform, p1);
01934 p2 = MatrixUtils.Multiply(_transform, p2);
01935 p3 = MatrixUtils.Multiply(_transform, p3);
01936 p4 = MatrixUtils.Multiply(_transform, p4);*/
01937
01938                         bounds = Point.Bounds(p1, p2, p3, p4);
01939
01940                         XmlElement maskElement = Document.CreateElement("mask", SVGNamespace);
01941                         maskElement.SetAttribute("id", filterGuid);
01942                         maskElement.SetAttribute("maskUnits", "userSpaceOnUse");
01943                         maskElement.SetAttribute("x",
      bounds.Location.X.ToString(System.Globalization.CultureInfo.InvariantCulture));
01944                         maskElement.SetAttribute("y",
      bounds.Location.Y.ToString(System.Globalization.CultureInfo.InvariantCulture));
01945                         maskElement.SetAttribute("width",
      bounds.Size.Width.ToString(System.Globalization.CultureInfo.InvariantCulture));
01946                         maskElement.SetAttribute("height",
      bounds.Size.Height.ToString(System.Globalization.CultureInfo.InvariantCulture));
01947
01948                         this.Definitions.AppendChild(maskElement);
01949
01950                         currentElement = maskElement;
01951
01952                         double[,] currTransform = _transform;
01953                         _transform = MatrixUtils.Identity;
01954
01955                         mask.Mask.CopyToIGraphicsContext(this);
01956
01957                         _transform = currTransform;
01958
01959                         currentElement = Document.CreateElement("g", SVGNamespace);
01960
01961                         if (!string.IsNullOrEmpty(this.Tag))
01962                         {
01963                             currentElement.SetAttribute("id", this.TagPrefix + Tag);
01964                         }
01965
01966                         currentElement.SetAttribute("mask", "url(#" + filterGuid + ")");
01967                         currentElement2.AppendChild(currentElement);
01968
01969                         currentElement.SetAttribute("transform", "matrix(" + _transform[0,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + _transform[1,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01970                     "," + _transform[0, 1].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
01971                     "," + _transform[0, 2].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
01972                         currTransform = _transform;
01973                         _transform = MatrixUtils.Identity;
01974
01975                         graphics.CopyToIGraphicsContext(this);
01976
01977                         _transform = currTransform;
01978
01979                         currentElement = currElement;
01980                     }
01981                     if (filter is GaussianBlurFilter gauss)
01982                     {
01983                         rasterisationNeeded = false;
01984                         justDraw = false;
01985                         XmlElement currElement = currentElement;
01986
01987                         if (!string.IsNullOrEmpty(_currClipPath))
01988                         {
01989                             currentElement = Document.CreateElement("g", SVGNamespace);
01990                             currentElement.SetAttribute("clip-path", _currClipPath);
01991                             currElement.AppendChild(currentElement);
01992                         }
01993
01994                         Rectangle bounds = graphics.GetBounds();
01995
01996                         Point p1 = new Point(bounds.Location.X - gauss.StandardDeviation * 3,
      bounds.Location.Y - gauss.StandardDeviation * 3);
01997                         Point p2 = new Point(bounds.Location.X + bounds.Size.Width +
      gauss.StandardDeviation * 3, bounds.Location.Y - gauss.StandardDeviation * 3);
```

```
01998                         Point p3 = new Point(bounds.Location.X + bounds.Size.Width +
      gauss.StandardDeviation * 3, bounds.Location.Y + bounds.Size.Height + gauss.StandardDeviation * 3);
01999                         Point p4 = new Point(bounds.Location.X - gauss.StandardDeviation * 3,
      bounds.Location.Y + bounds.Size.Height + gauss.StandardDeviation * 3);
02000
02001                         /*p1 = MatrixUtils.Multiply(_transform, p1);
02002 p2 = MatrixUtils.Multiply(_transform, p2);
02003 p3 = MatrixUtils.Multiply(_transform, p3);
02004 p4 = MatrixUtils.Multiply(_transform, p4);*/
02005
02006                         bounds = Point.Bounds(p1, p2, p3, p4);
02007
02008                         string filterGuid = !string.IsNullOrEmpty(Tag) ?  (this.TagPrefix + Tag +
      "@filter") :  Guid.NewGuid().ToString("N");
02009
02010                         XmlElement filterElement = Document.CreateElement("filter", SVGNamespace);
02011                         filterElement.SetAttribute("id", filterGuid);
02012                         filterElement.SetAttribute("color-interpolation-filters", "sRGB");
02013                         filterElement.SetAttribute("filterUnits", "userSpaceOnUse");
02014                         filterElement.SetAttribute("x",
      bounds.Location.X.ToString(System.Globalization.CultureInfo.InvariantCulture));
02015                         filterElement.SetAttribute("y",
      bounds.Location.Y.ToString(System.Globalization.CultureInfo.InvariantCulture));
02016                         filterElement.SetAttribute("width",
      bounds.Size.Width.ToString(System.Globalization.CultureInfo.InvariantCulture));
02017                         filterElement.SetAttribute("height",
      bounds.Size.Height.ToString(System.Globalization.CultureInfo.InvariantCulture));
02018
02019                         this.Definitions.AppendChild(filterElement);
02020
02021                         XmlElement feElement = Document.CreateElement("feGaussianBlur", SVGNamespace);
02022                         feElement.SetAttribute("stdDeviation",
      gauss.StandardDeviation.ToString(System.Globalization.CultureInfo.InvariantCulture));
02023
02024                         if (!string.IsNullOrEmpty(Tag))
02025                         {
02026                             feElement.SetAttribute("id", this.TagPrefix + Tag + "@feGaussianBlur");
02027                         }
02028
02029                         filterElement.AppendChild(feElement);
02030
02031                         XmlElement currentElement2 = currentElement;
02032
02033                         currentElement = Document.CreateElement("g", SVGNamespace);
02034
02035                         if (!string.IsNullOrEmpty(this.Tag))
02036                         {
02037                             currentElement.SetAttribute("id", this.TagPrefix + Tag);
02038                         }
02039
02040                         currentElement.SetAttribute("filter", "url(#" + filterGuid + ")");
02041                         currentElement2.AppendChild(currentElement);
02042
02043                         currentElement.SetAttribute("transform", "matrix(" + _transform[0,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + _transform[1,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
02044                                                     "," + _transform[0, 1].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
02045                                                     "," + _transform[0, 2].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
02046                         double[,] currTransform = _transform;
02047                         _transform = MatrixUtils.Identity;
02048
02049                         graphics.CopyToIGraphicsContext(this);
02050
02051                         currentElement = currElement;
02052                     }
02053                     else if (filter is ColourMatrixFilter cmf)
02054                     {
02055                         rasterisationNeeded = false;
02056                         justDraw = false;
02057                         XmlElement currElement = currentElement;
02058
02059                         if (!string.IsNullOrEmpty(_currClipPath))
02060                         {
02061                             currentElement = Document.CreateElement("g", SVGNamespace);
02062                             currentElement.SetAttribute("clip-path", _currClipPath);
02063                             currElement.AppendChild(currentElement);
02064                         }
02065
02066                         Rectangle bounds = graphics.GetBounds();
02067
02068                         Point p1 = new Point(bounds.Location.X, bounds.Location.Y);
02069                         Point p2 = new Point(bounds.Location.X + bounds.Size.Width,
      bounds.Location.Y);
02070                         Point p3 = new Point(bounds.Location.X + bounds.Size.Width, bounds.Location.Y
      + bounds.Size.Height);
```

```
02071                           Point p4 = new Point(bounds.Location.X, bounds.Location.Y +
      bounds.Size.Height);
02072
02073                           /* p1 = MatrixUtils.Multiply(_transform, p1);
02074 p2 = MatrixUtils.Multiply(_transform, p2);
02075 p3 = MatrixUtils.Multiply(_transform, p3);
02076 p4 = MatrixUtils.Multiply(_transform, p4);*/
02077
02078                           bounds = Point.Bounds(p1, p2, p3, p4);
02079
02080                           string filterGuid = !string.IsNullOrEmpty(Tag) ?  (this.TagPrefix + Tag +
      "@filter") :  Guid.NewGuid().ToString("N");
02081
02082                           XmlElement filterElement = Document.CreateElement("filter", SVGNamespace);
02083                           filterElement.SetAttribute("id", filterGuid);
02084                           filterElement.SetAttribute("color-interpolation-filters", "sRGB");
02085                           filterElement.SetAttribute("filterUnits", "userSpaceOnUse");
02086                           filterElement.SetAttribute("x",
      bounds.Location.X.ToString(System.Globalization.CultureInfo.InvariantCulture));
02087                           filterElement.SetAttribute("y",
      bounds.Location.Y.ToString(System.Globalization.CultureInfo.InvariantCulture));
02088                           filterElement.SetAttribute("width",
      bounds.Size.Width.ToString(System.Globalization.CultureInfo.InvariantCulture));
02089                           filterElement.SetAttribute("height",
      bounds.Size.Height.ToString(System.Globalization.CultureInfo.InvariantCulture));
02090
02091                           this.Definitions.AppendChild(filterElement);
02092
02093                           XmlElement feElement = Document.CreateElement("feColorMatrix", SVGNamespace);
02094                           feElement.SetAttribute("type", "matrix");
02095
02096                           if (!string.IsNullOrEmpty(Tag))
02097                           {
02098                               feElement.SetAttribute("id", this.TagPrefix + Tag + "@feColorMatrix");
02099                           }
02100
02101                           StringBuilder matrix = new StringBuilder();
02102
02103                           for (int i = 0; i < 4; i++)
02104                           {
02105                               for (int j = 0; j < 5; j++)
02106                               {
02107                                   matrix.Append(cmf.ColourMatrix[i,
      j].ToString(System.Globalization.CultureInfo.InvariantCulture));
02108                                   if (i != 3 || j != 4)
02109                                   {
02110                                       matrix.Append(" ");
02111                                   }
02112                               }
02113                           }
02114
02115                           feElement.SetAttribute("values", matrix.ToString());
02116                           filterElement.AppendChild(feElement);
02117
02118                           XmlElement currentElement2 = currentElement;
02119
02120                           currentElement = Document.CreateElement("g", SVGNamespace);
02121
02122                           if (!string.IsNullOrEmpty(this.Tag))
02123                           {
02124                               currentElement.SetAttribute("id", this.TagPrefix + Tag);
02125                           }
02126
02127                           currentElement.SetAttribute("filter", "url(#" + filterGuid + ")");
02128                           currentElement2.AppendChild(currentElement);
02129
02130                           currentElement.SetAttribute("transform", "matrix(" + _transform[0,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + _transform[1,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
02131                       "," + _transform[0, 1].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
02132                       "," + _transform[0, 2].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
02133                           double[,] currTransform = _transform;
02134                           _transform = MatrixUtils.Identity;
02135
02136                           graphics.CopyToIGraphicsContext(this);
02137
02138                           currentElement = currElement;
02139                       }
02140                       else if (filter is CompositeLocationInvariantFilter comp)
02141                       {
02142                           bool allSupported = true;
02143
02144                           foreach (IFilter filter2 in comp.Filters)
02145                           {
02146                               if (!(filter2 is GaussianBlurFilter) && !(filter2 is ColourMatrixFilter))
```

```
02147                           {
02148                               allSupported = false;
02149                               break;
02150                           }
02151                       }
02152 
02153                       if (allSupported)
02154                       {
02155                           rasterisationNeeded = false;
02156                           justDraw = false;
02157 
02158                           XmlElement currElement = currentElement;
02159 
02160                           if (!string.IsNullOrEmpty(_currClipPath))
02161                           {
02162                               currentElement = Document.CreateElement("g", SVGNamespace);
02163                               currentElement.SetAttribute("clip-path", _currClipPath);
02164                               currElement.AppendChild(currentElement);
02165                           }
02166 
02167                           Rectangle bounds = graphics.GetBounds();
02168 
02169                           Point p1 = new Point(bounds.Location.X - comp.TopLeftMargin.X,
    bounds.Location.Y - comp.TopLeftMargin.Y);
02170                           Point p2 = new Point(bounds.Location.X + bounds.Size.Width +
    comp.BottomRightMargin.X, bounds.Location.Y - comp.TopLeftMargin.Y);
02171                           Point p3 = new Point(bounds.Location.X + bounds.Size.Width +
    comp.BottomRightMargin.X, bounds.Location.Y + bounds.Size.Height + comp.BottomRightMargin.Y);
02172                           Point p4 = new Point(bounds.Location.X - comp.TopLeftMargin.X,
    bounds.Location.Y + bounds.Size.Height + comp.BottomRightMargin.Y);
02173 
02174                           /* p1 = MatrixUtils.Multiply(_transform, p1);
02175 p2 = MatrixUtils.Multiply(_transform, p2);
02176 p3 = MatrixUtils.Multiply(_transform, p3);
02177 p4 = MatrixUtils.Multiply(_transform, p4);*/
02178 
02179                           bounds = Point.Bounds(p1, p2, p3, p4);
02180 
02181                           string filterGuid = !string.IsNullOrEmpty(Tag) ?  (this.TagPrefix + Tag +
    "@filter") :  Guid.NewGuid().ToString("N");
02182 
02183                           XmlElement filterElement = Document.CreateElement("filter", SVGNamespace);
02184                           filterElement.SetAttribute("id", filterGuid);
02185                           filterElement.SetAttribute("color-interpolation-filters", "sRGB");
02186                           filterElement.SetAttribute("filterUnits", "userSpaceOnUse");
02187                           filterElement.SetAttribute("x",
    bounds.Location.X.ToString(System.Globalization.CultureInfo.InvariantCulture));
02188                           filterElement.SetAttribute("y",
    bounds.Location.Y.ToString(System.Globalization.CultureInfo.InvariantCulture));
02189                           filterElement.SetAttribute("width",
    bounds.Size.Width.ToString(System.Globalization.CultureInfo.InvariantCulture));
02190                           filterElement.SetAttribute("height",
    bounds.Size.Height.ToString(System.Globalization.CultureInfo.InvariantCulture));
02191 
02192                           this.Definitions.AppendChild(filterElement);
02193 
02194                           int index = 0;
02195 
02196                           foreach (IFilter filter2 in comp.Filters)
02197                           {
02198                               if (filter2 is GaussianBlurFilter gauss2)
02199                               {
02200                                   XmlElement feElement = Document.CreateElement("feGaussianBlur",
    SVGNamespace);
02201                                   feElement.SetAttribute("stdDeviation",
    gauss2.StandardDeviation.ToString(System.Globalization.CultureInfo.InvariantCulture));
02202 
02203                                   if (!string.IsNullOrEmpty(Tag))
02204                                   {
02205                                       filterElement.SetAttribute("id", this.TagPrefix + Tag +
    "@feGaussianBlur" + index.ToString());
02206                                   }
02207 
02208                                   filterElement.AppendChild(feElement);
02209                               }
02210                               else if (filter2 is ColourMatrixFilter cmf2)
02211                               {
02212                                   XmlElement feElement = Document.CreateElement("feColorMatrix",
    SVGNamespace);
02213                                   feElement.SetAttribute("type", "matrix");
02214                                   if (!string.IsNullOrEmpty(Tag))
02215                                   {
02216                                       filterElement.SetAttribute("id", this.TagPrefix + Tag +
    "@feColorMatrix" + index.ToString());
02217                                   }
02218 
02219                                   StringBuilder matrix = new StringBuilder();
```

```
02220
02221                                   for (int i = 0; i < 4; i++)
02222                                   {
02223                                       for (int j = 0; j < 5; j++)
02224                                       {
02225                                           matrix.Append(cmf2.ColourMatrix[i,
      j].ToString(System.Globalization.CultureInfo.InvariantCulture));
02226                                           if (i != 3 || j != 4)
02227                                           {
02228                                               matrix.Append(" ");
02229                                           }
02230                                       }
02231                                   }
02232
02233                                   feElement.SetAttribute("values", matrix.ToString());
02234                                   filterElement.AppendChild(feElement);
02235                               }
02236
02237                               index++;
02238                           }
02239
02240                           XmlElement currentElement2 = currentElement;
02241
02242                           currentElement = Document.CreateElement("g", SVGNamespace);
02243
02244                           if (!string.IsNullOrEmpty(this.Tag))
02245                           {
02246                               currentElement.SetAttribute("id", this.TagPrefix + Tag);
02247                           }
02248
02249                           currentElement.SetAttribute("filter", "url(#" + filterGuid + ")");
02250                           currentElement2.AppendChild(currentElement);
02251
02252                           currentElement.SetAttribute("transform", "matrix(" + _transform[0,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + _transform[1,
      0].ToString(System.Globalization.CultureInfo.InvariantCulture) +
02253                       "," + _transform[0, 1].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 1].ToString(System.Globalization.CultureInfo.InvariantCulture) +
02254                       "," + _transform[0, 2].ToString(System.Globalization.CultureInfo.InvariantCulture)
      + "," + _transform[1, 2].ToString(System.Globalization.CultureInfo.InvariantCulture) + ")");
02255                           double[,] currTransform = _transform;
02256                           _transform = MatrixUtils.Identity;
02257
02258                           graphics.CopyToIGraphicsContext(this);
02259
02260                           currentElement = currElement;
02261
02262
02263                       }
02264                   }
02265               }
02266
02267               if (rasterisationNeeded)
02268               {
02269                   double scale = FilterOption.RasterisationResolution;
02270
02271                   Rectangle bounds = graphics.GetBounds();
02272
02273                   bounds = new Rectangle(bounds.Location.X - filter.TopLeftMargin.X,
      bounds.Location.Y - filter.TopLeftMargin.Y, bounds.Size.Width + filter.TopLeftMargin.X +
      filter.BottomRightMargin.X, bounds.Size.Height + filter.TopLeftMargin.Y + filter.BottomRightMargin.Y);
02274
02275                   if (bounds.Size.Width > 0 && bounds.Size.Height > 0)
02276                   {
02277                       if (!FilterOption.RasterisationResolutionRelative)
02278                       {
02279                           scale = scale / Math.Min(bounds.Size.Width, bounds.Size.Height);
02280                       }
02281
02282                       if (graphics.TryRasterise(bounds, scale, true, out RasterImage rasterised))
02283                       {
02284                           RasterImage filtered = null;
02285
02286                           if (filter is ILocationInvariantFilter locInvFilter)
02287                           {
02288                               filtered = locInvFilter.Filter(rasterised, scale);
02289                           }
02290                           else if (filter is IFilterWithLocation filterWithLoc)
02291                           {
02292                               filtered = filterWithLoc.Filter(rasterised, bounds, scale);
02293                           }
02294
02295                           if (filtered != null)
02296                           {
02297                               rasterised.Dispose();
02298
02299                               DrawRasterImage(0, 0, filtered.Width, filtered.Height,
```

```
    bounds.Location.X, bounds.Location.Y, bounds.Size.Width, bounds.Size.Height, filtered);
02300                           }
02301                       }
02302                       else
02303                       {
02304                           throw new NotImplementedException(@"The filter could not be rasterised!
    You can avoid this error by doing one of the following:
02305 • Add a reference to VectSharp.Raster or VectSharp.Raster.ImageSharp (you may also need to add a using
    directive somewhere to force the assembly to be loaded).
02306 • Provide your own implementation of Graphics.RasterisationMethod.
02307 • Set the FilterOption.Operation to ""NeverRasteriseAndIgnore"", ""NeverRasteriseAndSkip"",
    ""IgnoreAll"" or ""SkipAll"".");
02308                       }
02309                   }
02310               }
02311
02312               if (justDraw)
02313               {
02314                   graphics.CopyToIGraphicsContext(this);
02315               }
02316           }
02317       }
02318   }
02319
02320
02321 /// <summary>
02322 /// Contains methods to render a <see cref="Page"/> as an SVG file.
02323 /// </summary>
02324   public static class SVGContextInterpreter
02325   {
02326
02327 /// <summary>
02328 /// Render the page to an SVG file.
02329 /// </summary>
02330 /// <param name="page">The <see cref="Page"/> to render.</param>
02331 /// <param name="fileName">The full path to the file to save.  If it exists, it will be
    overwritten.</param>
02332 /// <param name="textOption">Defines whether the used fonts should be included in the file.</param>
02333 /// <param name="linkDestinations">A dictionary associating element tags to link targets.  If this is
    provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to
    the destination specified in the dictionary.  If the destination starts with a hash (#), it is
    interpreted as the tag of another object in the current document; otherwise, it is interpreted as an
    external URI.</param>
02334 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
    rendering the image.</param>
02335 /// <param name="useStyles">If this is <see langword="false"/>, presentation attributes are set as
    attributes on SVG elements.  If this is <see langword="true"/>, CSS classes are used to set
    presentation attributes.</param>
02336       public static void SaveAsSVG(this Page page, string fileName, TextOptions textOption =
    TextOptions.SubsetFonts, Dictionary<string, string> linkDestinations = null, FilterOption filterOption
    = default, bool useStyles = false)
02337       {
02338           using (FileStream sr = new FileStream(fileName, FileMode.Create))
02339           {
02340               page.SaveAsSVG(sr, textOption, linkDestinations, filterOption, useStyles);
02341           }
02342       }
02343
02344 /// <summary>
02345 /// Defines whether the used fonts should be included in the file.
02346 /// </summary>
02347       public enum TextOptions
02348       {
02349 /// <summary>
02350 /// Embeds the full font files.
02351 /// </summary>
02352           EmbedFonts,
02353
02354 /// <summary>
02355 /// Embeds subsetted font files containing only the glyphs for the characters that have been used.
02356 /// </summary>
02357           SubsetFonts,
02358
02359 /// <summary>
02360 /// Does not embed any font file and converts all text items into paths.
02361 /// </summary>
02362           ConvertIntoPaths,
02363
02364 /// <summary>
02365 /// Does not embed any font file, but still encodes text items as such.
02366 /// </summary>
02367           DoNotEmbed,
02368
02369 /// <summary>
02370 /// Converts all text items into paths, but defines each glyph only once and reuses it when needed.
02371 /// </summary>
02372           ConvertIntoPathsUsingGlyphs
```

```
02373         }
02374
02375 /// <summary>
02376 /// Determines how and whether image filters are rasterised.
02377 /// </summary>
02378         public class FilterOption
02379         {
02380 /// <summary>
02381 /// Defines whether image filters should be rasterised or not.
02382 /// </summary>
02383             public enum FilterOperations
02384             {
02385 /// <summary>
02386 /// Image filters will always be rasterised.
02387 /// </summary>
02388                 RasteriseAll,
02389
02390 /// <summary>
02391 /// Image filters will only be rasterised if they are not supported natively by the output file
      format.
02392 /// </summary>
02393                 RasteriseIfNecessary,
02394
02395 /// <summary>
02396 /// Image filters will never be rasterised; for filters that are not supported, the filter will be
      ignored.
02397 /// </summary>
02398                 NeverRasteriseAndIgnore,
02399
02400 /// <summary>
02401 /// Image filters will never be rasterised; if an image should be drawn with an unsupported filter,
      the image will not be drawn at all.
02402 /// </summary>
02403                 NeverRasteriseAndSkip,
02404
02405 /// <summary>
02406 /// All image filters (supported and unsupported) will be ignored.
02407 /// </summary>
02408                 IgnoreAll,
02409
02410 /// <summary>
02411 /// All the images that should be drawn with a filter will be ignored.
02412 /// </summary>
02413                 SkipAll
02414             }
02415
02416 /// <summary>
02417 /// Defines whether image filters should be rasterised or not.
02418 /// </summary>
02419             public FilterOperations Operation { get; } = FilterOperations.RasteriseIfNecessary;
02420
02421 /// <summary>
02422 /// The resolution that will be used to rasterise image filters.  Depending on the value of <see
      cref="RasterisationResolutionRelative"/>, this can either be an absolute resolution (i.e.  a size in
      pixel), or a scale factor that is applied to the image size in graphics units.
02423 /// </summary>
02424             public double RasterisationResolution { get; } = 1;
02425
02426 /// <summary>
02427 /// Determines whether the value of <see cref="RasterisationResolution"/> is absolute (i.e.  a size in
      pixel), or relative (i.e.  a scale factor that is applied to the image size in graphics units).
02428 /// </summary>
02429             public bool RasterisationResolutionRelative { get; } = true;
02430
02431 /// <summary>
02432 /// The default options for image filter rasterisation.
02433 /// </summary>
02434             public static FilterOption Default = new
      FilterOption(FilterOperations.RasteriseIfNecessary, 1, true);
02435
02436 /// <summary>
02437 /// Create a new <see cref="FilterOption"/> object.
02438 /// </summary>
02439 /// <param name="operation">Defines whether image filters should be rasterised or not.</param>
02440 /// <param name="rasterisationResolution">The resolution that will be used to rasterise image filters.
      Depending on the value of <see cref="RasterisationResolutionRelative"/>, this can either be an
      absolute resolution (i.e.  a size in pixel), or a scale factor that is applied to the image size in
      graphics units.</param>
02441 /// <param name="rasterisationResolutionRelative">Determines whether the value of <see
      cref="RasterisationResolution"/> is absolute (i.e.  a size in pixel), or relative (i.e.  a scale
      factor that is applied to the image size in graphics units).</param>
02442             public FilterOption(FilterOperations operation, double rasterisationResolution, bool
      rasterisationResolutionRelative)
02443             {
02444                 this.Operation = operation;
02445                 this.RasterisationResolution = rasterisationResolution;
02446                 this.RasterisationResolutionRelative = rasterisationResolutionRelative;
```

```
02447             }
02448         }
02449 
02450 /// <summary>
02451 /// Render the page to an SVG stream.
02452 /// </summary>
02453 /// <param name="page">The <see cref="Page"/> to render.</param>
02454 /// <param name="stream">The stream to which the SVG data will be written.</param>
02455 /// <param name="textOption">Defines whether the used fonts should be included in the file.</param>
02456 /// <param name="linkDestinations">A dictionary associating element tags to link targets.  If this is
      provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to
      the destination specified in the dictionary.  If the destination starts with a hash (#), it is
      interpreted as the tag of another object in the current document; otherwise, it is interpreted as an
      external URI.</param>
02457 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
02458 /// <param name="useStyles">If this is <see langword="false"/>, presentation attributes are set as
      attributes on SVG elements.  If this is <see langword="true"/>, CSS classes are used to set
      presentation attributes.</param>
02459         public static void SaveAsSVG(this Page page, Stream stream, TextOptions textOption =
      TextOptions.SubsetFonts, Dictionary<string, string> linkDestinations = null, FilterOption filterOption
      = default, bool useStyles = false)
02460         {
02461             XmlDocument doc = page.SaveAsSVG(textOption, linkDestinations, filterOption, useStyles);
02462             WriteXMLToStream(doc.DocumentElement, stream);
02463         }
02464 
02465 /// <summary>
02466 /// Render the page to an SVG document.
02467 /// </summary>
02468 /// <param name="page">The <see cref="Page"/> to render.</param>
02469 /// <param name="textOption">Defines whether the used fonts should be included in the file.</param>
02470 /// <param name="linkDestinations">A dictionary associating element tags to link targets.  If this is
      provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to
      the destination specified in the dictionary.  If the destination starts with a hash (#), it is
      interpreted as the tag of another object in the current document; otherwise, it is interpreted as an
      external URI.</param>
02471 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
02472 /// <param name="useStyles">If this is <see langword="false"/>, presentation attributes are set as
      attributes on SVG elements.  If this is <see langword="true"/>, CSS classes are used to set
      presentation attributes.</param>
02473 /// <returns>An <see cref="XmlDocument"/> containing the rendered SVG image.</returns>
02474         public static XmlDocument SaveAsSVG(this Page page, TextOptions textOption =
      TextOptions.SubsetFonts, Dictionary<string, string> linkDestinations = null, FilterOption filterOption
      = default, bool useStyles = false)
02475         {
02476             return CreateSVGDocument(page, "", textOption, linkDestinations, filterOption, useStyles,
      true, new Dictionary<string, RasterImage>(), new Dictionary<string, (string, FontFamily,
      HashSet<char>)>(), true, new object());
02477         }
02478 
02479         private static XmlDocument CreateSVGDocument(this Page page, string tagPrefix, TextOptions
      textOption, Dictionary<string, string> linkDestinations, FilterOption filterOption, bool useStyles,
      bool reuseGradients, Dictionary<string, RasterImage> usedRasterImages, Dictionary<string, (string,
      FontFamily, HashSet<char>)> usedFonts, bool embedEverythingHere, object lockObject)
02480         {
02481             if (linkDestinations == null)
02482             {
02483                 linkDestinations = new Dictionary<string, string>();
02484             }
02485 
02486             if (filterOption == null)
02487             {
02488                 filterOption = FilterOption.Default;
02489             }
02490 
02491             bool textToPaths = textOption == TextOptions.ConvertIntoPaths;
02492 
02493             SVGContext ctx = new SVGContext(page.Width, page.Height, textToPaths, textOption,
      linkDestinations, filterOption, useStyles, tagPrefix, reuseGradients, usedRasterImages, lockObject);
02494 
02495             ctx.Rectangle(0, 0, page.Width, page.Height);
02496             ctx.SetFillStyle(page.Background);
02497             ctx.Fill(FillRule.NonZeroWinding);
02498             ctx.SetFillStyle((0, 0, 0, 1));
02499 
02500             page.Graphics.CopyToIGraphicsContext(ctx);
02501 
02502             if (!textToPaths && textOption != TextOptions.DoNotEmbed && textOption !=
      TextOptions.ConvertIntoPathsUsingGlyphs)
02503             {
02504                 bool subsetFonts = textOption == TextOptions.SubsetFonts;
02505 
02506                 Dictionary<string, string> newFontFamilies = new Dictionary<string, string>();
02507 
02508                 if (embedEverythingHere)
```

```
02509                 {
02510                     StringBuilder cssFonts = new StringBuilder();
02511 
02512                     foreach (KeyValuePair<string, FontFamily> kvp in ctx.UsedFontFamilies)
02513                     {
02514                         TrueTypeFile subsettedFont;
02515 
02516                         if (subsetFonts)
02517                         {
02518                             newFontFamilies[kvp.Key] = kvp.Value.TrueTypeFile.GetFontFamilyName() +
      "-" + Guid.NewGuid().ToString();
02519                             subsettedFont = kvp.Value.TrueTypeFile.SubsetFont(new
      string(ctx.UsedChars[kvp.Key].ToArray()));
02520                         }
02521                         else
02522                         {
02523                             newFontFamilies[kvp.Key] = kvp.Value.TrueTypeFile.GetFontName();
02524                             subsettedFont = kvp.Value.TrueTypeFile;
02525                         }
02526 
02527                         byte[] fontBytes;
02528 
02529                         using (MemoryStream fontStream = new
      MemoryStream((int)subsettedFont.FontStream.Length))
02530                         {
02531                             subsettedFont.FontStream.Seek(0, SeekOrigin.Begin);
02532                             subsettedFont.FontStream.CopyTo(fontStream);
02533 
02534                             fontBytes = fontStream.ToArray();
02535                         }
02536 
02537 
02538                         cssFonts.Append("\n    @font-face\n    {\n      font-family:  \"" +
      newFontFamilies[kvp.Key] + "\";\n      src:  url(\"data:font/ttf;charset=utf-8base64,");
02539                         cssFonts.Append(Convert.ToBase64String(fontBytes));
02540                         cssFonts.Append("\");\n    }\n  ");
02541                     }
02542 
02543                     XmlElement style = ctx.Document.CreateElement("style", SVGContext.SVGNamespace);
02544                     style.InnerText = cssFonts.ToString();
02545 
02546                     XmlNode svgElement = ctx.Document.GetElementsByTagName("svg")[0];
02547 
02548                     svgElement.InsertBefore(style, svgElement.FirstChild);
02549                 }
02550                 else
02551                 {
02552                     foreach (KeyValuePair<string, FontFamily> kvp in ctx.UsedFontFamilies)
02553                     {
02554                         string fName = kvp.Key;
02555 
02556                         lock (lockObject)
02557                         {
02558                             if (!usedFonts.TryGetValue(fName, out (string newName, FontFamily family,
      HashSet<char> usedChars) previous))
02559                             {
02560                                 string newName = kvp.Value.TrueTypeFile.GetFontFamilyName() + "-" +
      Guid.NewGuid().ToString();
02561                                 previous = (newName, kvp.Value, new HashSet<char>());
02562                                 usedFonts[fName] = previous;
02563                             }
02564                             else if (previous.family.TrueTypeFile != kvp.Value.TrueTypeFile)
02565                             {
02566                                 string transferredName = Guid.NewGuid().ToString();
02567                                 usedFonts[transferredName] = usedFonts[fName];
02568 
02569                                 string newName = kvp.Value.TrueTypeFile.GetFontFamilyName() + "-" +
      Guid.NewGuid().ToString();
02570                                 previous = (newName, kvp.Value, new HashSet<char>());
02571                                 usedFonts[fName] = previous;
02572                             }
02573 
02574                             previous.usedChars.UnionWith(ctx.UsedChars[fName]);
02575 
02576                             if (subsetFonts)
02577                             {
02578                                 newFontFamilies[fName] = previous.newName;
02579                             }
02580                             else
02581                             {
02582                                 newFontFamilies[fName] = previous.family.TrueTypeFile.GetFontName();
02583                             }
02584                         }
02585                     }
02586                 }
02587 
02588                 HashSet<string> updatedClasses = new HashSet<string>();
```

```
02589
02590                 foreach (XmlNode text in ctx.Document.GetElementsByTagName("text"))
02591                 {
02592                     string fontFamily;
02593
02594                     if (!useStyles)
02595                     {
02596                         fontFamily = text.Attributes["font-family"].Value;
02597                     }
02598                     else
02599                     {
02600                         fontFamily = ctx.Styles[text.Attributes["class"].Value]["font-family"];
02601                     }
02602
02603                     if (!useStyles || updatedClasses.Add(text.Attributes["class"].Value))
02604                     {
02605
02606                         string fallbackFontFamily = "";
02607
02608                         switch (fontFamily)
02609                         {
02610                             case "Helvetica":
02611                             case "Helvetica-Bold":
02612                             case "Helvetica-Oblique":
02613                             case "Helvetica-BoldOblique":
02614                                 fallbackFontFamily = "sans-serif";
02615                                 break;
02616
02617                             case "Times-Roman":
02618                             case "Times-Bold":
02619                             case "Times-Italic":
02620                             case "Times-BoldItalic":
02621                                 fallbackFontFamily = "serif";
02622                                 break;
02623
02624                             case "Courier":
02625                             case "Courier-Bold":
02626                             case "Courier-Oblique":
02627                             case "Courier-BoldOblique":
02628                                 fallbackFontFamily = "monospace";
02629                                 break;
02630
02631                             default:
02632                                 if (ctx.UsedFontFamilies[fontFamily].TrueTypeFile.IsFixedPitch())
02633                                 {
02634                                     fallbackFontFamily = "monospace";
02635                                 }
02636                                 else if (ctx.UsedFontFamilies[fontFamily].TrueTypeFile.IsScript() ||
      ctx.UsedFontFamilies[fontFamily].TrueTypeFile.IsSerif())
02637                                 {
02638                                     fallbackFontFamily = "serif";
02639                                 }
02640                                 else
02641                                 {
02642                                     fallbackFontFamily = "sans-serif";
02643                                 }
02644                                 break;
02645                         }
02646
02647                         if (!useStyles)
02648                         {
02649                             if (!string.IsNullOrEmpty(fallbackFontFamily))
02650                             {
02651                                 ((XmlElement)text).SetAttribute("font-family",
      newFontFamilies[fontFamily] + ", " + fallbackFontFamily);
02652                             }
02653                             else
02654                             {
02655                                 ((XmlElement)text).SetAttribute("font-family",
      newFontFamilies[fontFamily]);
02656                             }
02657                         }
02658                         else
02659                         {
02660                             if (!string.IsNullOrEmpty(fallbackFontFamily))
02661                             {
02662                                 ctx.Styles[text.Attributes["class"].Value]["font-family"] =
      newFontFamilies[fontFamily] + ", " + fallbackFontFamily;
02663                             }
02664                             else
02665                             {
02666                                 ctx.Styles[text.Attributes["class"].Value]["font-family"] =
      newFontFamilies[fontFamily];
02667                             }
02668                         }
02669                     }
02670                 }
```

```
02671             }
02672             else if (textOption == TextOptions.ConvertIntoPathsUsingGlyphs)
02673             {
02674                 if (embedEverythingHere)
02675                 {
02676                     foreach (KeyValuePair<string, FontFamily> kvp in ctx.UsedFontFamilies)
02677                     {
02678                         string guid = Guid.NewGuid().ToString();
02679 
02680                         XmlElement defs = ctx.Document.CreateElement("defs", SVGContext.SVGNamespace);
02681                         defs.SetAttribute("id", kvp.Value.FamilyName + "-glyphs-" + guid);
02682 
02683                         foreach (char c in ctx.UsedChars[kvp.Key])
02684                         {
02685                             XmlElement charPath = ctx.Document.CreateElement("path",
      SVGContext.SVGNamespace);
02686                             charPath.SetAttribute("id", kvp.Key + "-" + c);
02687 
02688                             TrueTypeFile.TrueTypePoint[][] glyphPaths =
      kvp.Value.TrueTypeFile.GetGlyphPath(c, kvp.Value.TrueTypeFile.GetUnitsPerEm());
02689 
02690                             StringBuilder data = new StringBuilder();
02691 
02692                             for (int j = 0; j < glyphPaths.Length; j++)
02693                             {
02694                                 for (int k = 0; k < glyphPaths[j].Length; k++)
02695                                 {
02696                                     if (k == 0)
02697                                     {
02698                                         data.Append("M " +
      glyphPaths[j][k].X.ToString(System.Globalization.CultureInfo.InvariantCulture) + "," +
      (-glyphPaths[j][k].Y).ToString(System.Globalization.CultureInfo.InvariantCulture) + " ");
02699                                     }
02700                                     else
02701                                     {
02702                                         if (glyphPaths[j][k].IsOnCurve)
02703                                         {
02704                                             data.Append("L " +
      glyphPaths[j][k].X.ToString(System.Globalization.CultureInfo.InvariantCulture) + "," +
      (-glyphPaths[j][k].Y).ToString(System.Globalization.CultureInfo.InvariantCulture) + " ");
02705                                         }
02706                                         else
02707                                         {
02708                                             data.Append("Q " +
      glyphPaths[j][k].X.ToString(System.Globalization.CultureInfo.InvariantCulture) + "," +
      (-glyphPaths[j][k].Y).ToString(System.Globalization.CultureInfo.InvariantCulture) + " ");
02709                                             data.Append(glyphPaths[j][k +
      1].X.ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + (-glyphPaths[j][k +
      1].Y).ToString(System.Globalization.CultureInfo.InvariantCulture) + " ");
02710 
02711                                             k++;
02712                                         }
02713                                     }
02714                                 }
02715 
02716                                 data.Append("Z");
02717                             }
02718 
02719                             charPath.SetAttribute("d", data.ToString());
02720 
02721                             defs.AppendChild(charPath);
02722                         }
02723 
02724                         XmlNode svgElement = ctx.Document.GetElementsByTagName("svg")[0];
02725                         svgElement.InsertBefore(defs, svgElement.FirstChild);
02726                     }
02727                 }
02728                 else
02729                 {
02730                     foreach (KeyValuePair<string, FontFamily> kvp in ctx.UsedFontFamilies)
02731                     {
02732                         string fName = kvp.Key;
02733 
02734                         lock (lockObject)
02735                         {
02736                             if (!usedFonts.TryGetValue(fName, out (string newName, FontFamily family,
      HashSet<char> usedChars) previous))
02737                             {
02738                                 string newName = kvp.Value.TrueTypeFile.GetFontFamilyName() + "-" +
      Guid.NewGuid().ToString();
02739                                 previous = (newName, kvp.Value, new HashSet<char>());
02740                                 usedFonts[fName] = previous;
02741                             }
02742                             else if (previous.family.TrueTypeFile != kvp.Value.TrueTypeFile)
02743                             {
02744                                 string transferredName = Guid.NewGuid().ToString();
02745                                 usedFonts[transferredName] = usedFonts[fName];
```

```
02746
02747                                   string newName = kvp.Value.TrueTypeFile.GetFontFamilyName() + "-" +
     Guid.NewGuid().ToString();
02748                                   previous = (newName, kvp.Value, new HashSet<char>());
02749                                   usedFonts[fName] = previous;
02750                               }
02751
02752                               previous.usedChars.UnionWith(ctx.UsedChars[fName]);
02753                           }
02754                       }
02755                   }
02756               }
02757               else if (!textToPaths && textOption == TextOptions.DoNotEmbed)
02758               {
02759                   HashSet<string> updatedClasses = new HashSet<string>();
02760
02761                   foreach (XmlNode text in ctx.Document.GetElementsByTagName("text"))
02762                   {
02763                       string fontFamily;
02764
02765                       if (!useStyles)
02766                       {
02767                           fontFamily = text.Attributes["font-family"].Value;
02768                       }
02769                       else
02770                       {
02771                           fontFamily = ctx.Styles[text.Attributes["class"].Value]["font-family"];
02772                       }
02773
02774                       if (!useStyles || updatedClasses.Add(text.Attributes["class"].Value))
02775                       {
02776
02777                           string newFontFamily =
     ctx.UsedFontFamilies[fontFamily].TrueTypeFile.GetFontFamilyName();
02778
02779                           switch (fontFamily)
02780                           {
02781                               case "Helvetica":
02782                               case "Helvetica-Bold":
02783                               case "Helvetica-Oblique":
02784                               case "Helvetica-BoldOblique":
02785                                   newFontFamily = newFontFamily + ", sans-serif";
02786                                   break;
02787
02788                               case "Times-Roman":
02789                               case "Times-Bold":
02790                               case "Times-Italic":
02791                               case "Times-BoldItalic":
02792                                   newFontFamily = newFontFamily + ", serif";
02793                                   break;
02794
02795                               case "Courier":
02796                               case "Courier-Bold":
02797                               case "Courier-Oblique":
02798                               case "Courier-BoldOblique":
02799                                   newFontFamily = newFontFamily + ", monospace";
02800                                   break;
02801
02802                               default:
02803                                   if (ctx.UsedFontFamilies[fontFamily].TrueTypeFile.IsFixedPitch())
02804                                   {
02805                                       newFontFamily = newFontFamily + ", monospace";
02806                                   }
02807                                   else if (ctx.UsedFontFamilies[fontFamily].TrueTypeFile.IsScript())
02808                                   {
02809                                       newFontFamily = newFontFamily + ", cursive";
02810                                   }
02811                                   else if (ctx.UsedFontFamilies[fontFamily].TrueTypeFile.IsSerif())
02812                                   {
02813                                       newFontFamily = newFontFamily + ", serif";
02814                                   }
02815                                   else
02816                                   {
02817                                       newFontFamily = newFontFamily + ", sans-serif";
02818                                   }
02819                                   break;
02820                           }
02821
02822                           if (!useStyles)
02823                           {
02824                               ((XmlElement)text).SetAttribute("font-family", newFontFamily);
02825                           }
02826                           else
02827                           {
02828                               ctx.Styles[text.Attributes["class"].Value]["font-family"] = newFontFamily;
02829                           }
02830                       }
```

```
02831                 }
02832             }
02833 
02834             if (useStyles)
02835             {
02836                 StringBuilder classStyle = new StringBuilder();
02837 
02838                 if (ctx.Styles.Count > 0)
02839                 {
02840                     classStyle.Append("\n  ");
02841                 }
02842 
02843                 foreach (KeyValuePair<string, Dictionary<string, string» style in ctx.Styles)
02844                 {
02845                     classStyle.Append("  .");
02846                     classStyle.Append(style.Key);
02847                     classStyle.Append("\n");
02848                     classStyle.Append("    {\n");
02849 
02850                     foreach (KeyValuePair<string, string> kvp in style.Value)
02851                     {
02852                         classStyle.Append("      ");
02853                         classStyle.Append(kvp.Key);
02854                         classStyle.Append(":  ");
02855                         classStyle.Append(kvp.Value);
02856                         classStyle.Append(";\n");
02857                     }
02858 
02859                     classStyle.Append("    }\n  ");
02860                 }
02861 
02862 
02863                 XmlElement styleElement = ctx.Document.CreateElement("style",
      SVGContext.SVGNamespace);
02864                 styleElement.InnerText = classStyle.ToString();
02865 
02866                 XmlNode svgElement = ctx.Document.GetElementsByTagName("svg")[0];
02867 
02868                 svgElement.InsertBefore(styleElement, svgElement.FirstChild);
02869             }
02870 
02871             if (embedEverythingHere && usedRasterImages.Count > 0)
02872             {
02873                 foreach (KeyValuePair<string, RasterImage> kvp in usedRasterImages)
02874                 {
02875                     EmbedRasterImage(ctx.Document, kvp.Key, kvp.Value, ctx.Definitions);
02876                 }
02877             }
02878 
02879             ctx.Document.DocumentElement.SetAttribute("style", "font-synthesis:  none;");
02880 
02881             return ctx.Document;
02882         }
02883 
02884         private static void EmbedRasterImage(XmlDocument document, string id, RasterImage image,
      XmlElement definitions)
02885         {
02886             XmlElement img = document.CreateElement("image", SVGContext.SVGNamespace);
02887             img.SetAttribute("id", id);
02888             img.SetAttribute("x", "0");
02889             img.SetAttribute("y", "0");
02890 
02891             img.SetAttribute("width", "1");
02892             img.SetAttribute("height", "1");
02893 
02894             img.SetAttribute("preserveAspectRatio", "none");
02895 
02896             if (image.Interpolate)
02897             {
02898                 img.SetAttribute("image-rendering", "optimizeQuality");
02899             }
02900             else
02901             {
02902                 img.SetAttribute("image-rendering", "pixelated");
02903             }
02904 
02905             img.SetAttribute("href", "http://www.w3.org/1999/xlink", "" +
      Convert.ToBase64String(image.PNGStream.ToArray()));
02906 
02907             definitions.AppendChild(img);
02908         }
02909 
02910 /// <summary>
02911 /// Render the animation to an SVG document, using SVG animations.
02912 /// </summary>
02913 /// <param name="animation">The <see cref="Animation"/> to render.</param>
02914 /// <param name="includeControls">If this is <see langword="true"/>, the generated SVG file will
```

```
contain playback controls that use Javascript to play/pause the animation and change the current time.</param>
02915 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note that this does not affect the frame rate of the animation.</param>
02916 /// <param name="textOption">Defines whether the used fonts should be included in the file.</param>
02917 /// <param name="linkDestinations">A dictionary associating element tags to link targets.  If this is provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to the destination specified in the dictionary.  If the destination starts with a hash (#), it is interpreted as the tag of another object in the current document; otherwise, it is interpreted as an external URI.</param>
02918 /// <param name="filterOption">Defines how and whether image filters should be rasterised when rendering the image.</param>
02919 /// <returns>An <see cref="XmlDocument"/> containing the animated SVG image.</returns>
02920         public static XmlDocument SaveAsAnimatedSVG(this Animation animation, bool includeControls = false, double durationScaling = 1, TextOptions textOption = TextOptions.SubsetFonts, Dictionary<string, string> linkDestinations = null, FilterOption filterOption = default)
02921         {
02922             (XmlDocument, double)[] frames = new (XmlDocument, double)[animation.Frames.Count];
02923             double[] durations = new double[animation.Frames.Count];
02924 
02925             (XmlDocument, XmlDocument, Transition)[] transitions = new (XmlDocument, XmlDocument, Transition)[animation.Frames.Count - 1];
02926             Dictionary<string, RasterImage> usedRasterImages = new Dictionary<string, RasterImage>();
02927             Dictionary<string, (string, FontFamily, HashSet<char>)> usedFonts = new Dictionary<string, (string, FontFamily, HashSet<char>)>();
02928 
02929             string repeatCount = animation.RepeatCount <= 0 ?  "indefinite" : animation.RepeatCount.ToString();
02930 
02931             double currentTime = 0;
02932 
02933             for (int i = 0; i < frames.Length; i++)
02934             {
02935                 Page pag = new Page(animation.Width, animation.Height) { Background = animation.Background };
02936                 pag.Graphics.DrawGraphics(0, 0, animation.Frames[i].Graphics);
02937 
02938                 if (i > 0)
02939                 {
02940                     frames[i] = (pag.CreateSVGDocument("frame" + i.ToString() + "://", textOption, linkDestinations, filterOption, false, false, usedRasterImages, usedFonts, false, new object()), animation.Frames[i].Duration);
02941                     durations[i] = animation.Frames[i].Duration * durationScaling;
02942 
02943                     if (animation.Transitions[i - 1].Duration > 0)
02944                     {
02945                         double epsilon = 1e-5 * animation.Transitions[i - 1].Duration;
02946 
02947                         Page startPage = animation.GetFrameAtAbsolute(currentTime + epsilon);
02948                         Page endPage = animation.GetFrameAtAbsolute(currentTime + animation.Transitions[i - 1].Duration - epsilon);
02949 
02950                         transitions[i - 1] = (startPage.CreateSVGDocument("transition" + i.ToString() + "://", textOption, linkDestinations, filterOption, false, false, usedRasterImages, usedFonts, false, new object()), endPage.CreateSVGDocument("transition" + i.ToString() + "://", textOption, linkDestinations, filterOption, false, false, usedRasterImages, usedFonts, false, new object()), animation.Transitions[i - 1]);
02951                     }
02952                     else
02953                     {
02954                         transitions[i - 1] = (null, null, animation.Transitions[i - 1]);
02955                     }
02956 
02957                     currentTime += animation.Frames[i].Duration + animation.Transitions[i - 1].Duration;
02958                 }
02959                 else
02960                 {
02961                     frames[i] = (pag.CreateSVGDocument("frame" + i.ToString() + "://", textOption, linkDestinations, filterOption, false, false, usedRasterImages, usedFonts, false, new object()), animation.Frames[i].Duration);
02962                     durations[i] = animation.Frames[i].Duration * durationScaling;
02963                     currentTime += animation.Frames[i].Duration;
02964                 }
02965             }
02966 
02967             double totalDuration = animation.Duration * durationScaling;
02968 
02969             XmlDocument Document = new XmlDocument();
02970 
02971             Document.InsertBefore(Document.CreateXmlDeclaration("1.0", "UTF-8", null), Document.DocumentElement);
02972 
02973             XmlElement currentElement = Document.CreateElement(null, "svg", SVGContext.SVGNamespace);
02974             currentElement.SetAttribute("xmlns:xlink", "http://www.w3.org/1999/xlink");
02975             currentElement.SetAttribute("viewBox", "0 0 " +
```

```
      animation.Width.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      animation.Height.ToString(System.Globalization.CultureInfo.InvariantCulture));
02976             currentElement.SetAttribute("version", "1.1");
02977             Document.AppendChild(currentElement);
02978 
02979             if (usedFonts.Count > 0 && (textOption == TextOptions.SubsetFonts || textOption ==
      TextOptions.EmbedFonts))
02980             {
02981                 StringBuilder cssFonts = new StringBuilder();
02982 
02983                 foreach (KeyValuePair<string, (string, FontFamily, HashSet<char>)> kvp in usedFonts)
02984                 {
02985                     TrueTypeFile subsettedFont;
02986 
02987                     if (textOption == TextOptions.SubsetFonts)
02988                     {
02989                         subsettedFont = kvp.Value.Item2.TrueTypeFile.SubsetFont(new
      string(kvp.Value.Item3.ToArray()));
02990                     }
02991                     else
02992                     {
02993                         subsettedFont = kvp.Value.Item2.TrueTypeFile;
02994                     }
02995 
02996                     byte[] fontBytes;
02997 
02998                     using (MemoryStream fontStream = new
      MemoryStream((int)subsettedFont.FontStream.Length))
02999                     {
03000                         subsettedFont.FontStream.Seek(0, SeekOrigin.Begin);
03001                         subsettedFont.FontStream.CopyTo(fontStream);
03002 
03003                         fontBytes = fontStream.ToArray();
03004                     }
03005 
03006 
03007                     cssFonts.Append("\n    @font-face\n    {\n      font-family:  \"" +
      kvp.Value.Item1 + "\";\n      src:  url(\"data:font/ttf;charset=utf-8base64,");
03008                     cssFonts.Append(Convert.ToBase64String(fontBytes));
03009                     cssFonts.Append("\");\n    }\n  ");
03010                 }
03011 
03012                 XmlElement style = Document.CreateElement("style", SVGContext.SVGNamespace);
03013                 style.InnerText = cssFonts.ToString();
03014                 currentElement.AppendChild(style);
03015             }
03016 
03017             if (usedFonts.Count > 0 && textOption == TextOptions.ConvertIntoPathsUsingGlyphs)
03018             {
03019                 foreach (KeyValuePair<string, (string, FontFamily, HashSet<char>)> kvp in usedFonts)
03020                 {
03021                     string guid = Guid.NewGuid().ToString();
03022 
03023                     XmlElement defs = Document.CreateElement("defs", SVGContext.SVGNamespace);
03024                     defs.SetAttribute("id", kvp.Value.Item2.FamilyName + "-glyphs-" + guid);
03025 
03026                     foreach (char c in kvp.Value.Item3)
03027                     {
03028                         XmlElement charPath = Document.CreateElement("path", SVGContext.SVGNamespace);
03029                         charPath.SetAttribute("id", kvp.Key + "-" + c);
03030 
03031                         TrueTypeFile.TrueTypePoint[][] glyphPaths =
      kvp.Value.Item2.TrueTypeFile.GetGlyphPath(c, kvp.Value.Item2.TrueTypeFile.GetUnitsPerEm());
03032 
03033                         StringBuilder data = new StringBuilder();
03034 
03035                         for (int j = 0; j < glyphPaths.Length; j++)
03036                         {
03037                             for (int k = 0; k < glyphPaths[j].Length; k++)
03038                             {
03039                                 if (k == 0)
03040                                 {
03041                                     data.Append("M " +
      glyphPaths[j][k].X.ToString(System.Globalization.CultureInfo.InvariantCulture) + "," +
      (-glyphPaths[j][k].Y).ToString(System.Globalization.CultureInfo.InvariantCulture) + " ");
03042                                 }
03043                                 else
03044                                 {
03045                                     if (glyphPaths[j][k].IsOnCurve)
03046                                     {
03047                                         data.Append("L " +
      glyphPaths[j][k].X.ToString(System.Globalization.CultureInfo.InvariantCulture) + "," +
      (-glyphPaths[j][k].Y).ToString(System.Globalization.CultureInfo.InvariantCulture) + " ");
03048                                     }
03049                                     else
03050                                     {
03051                                         data.Append("Q " +
```

```
    glyphPaths[j][k].X.ToString(System.Globalization.CultureInfo.InvariantCulture) + "," +
    (-glyphPaths[j][k].Y).ToString(System.Globalization.CultureInfo.InvariantCulture) + " ");
03052                                   data.Append(glyphPaths[j][k +
    1].X.ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + (-glyphPaths[j][k +
    1].Y).ToString(System.Globalization.CultureInfo.InvariantCulture) + " ");
03053
03054                                   k++;
03055                               }
03056                           }
03057                       }
03058
03059                       data.Append("Z");
03060                   }
03061
03062                   charPath.SetAttribute("d", data.ToString());
03063
03064                   defs.AppendChild(charPath);
03065               }
03066
03067               currentElement.AppendChild(defs);
03068           }
03069       }
03070
03071       if (usedRasterImages.Count > 0)
03072       {
03073           XmlElement definitions = Document.CreateElement("defs", SVGContext.SVGNamespace);
03074           currentElement.AppendChild(definitions);
03075
03076           foreach (KeyValuePair<string, RasterImage> kvp in usedRasterImages)
03077           {
03078               EmbedRasterImage(Document, kvp.Key, kvp.Value, definitions);
03079           }
03080       }
03081
03082       currentTime = 0;
03083
03084       for (int i = 0; i < frames.Length; i++)
03085       {
03086           if (i > 0 && transitions[i - 1].Item3.Duration > 0)
03087           {
03088               XmlNode clonedSVG = Document.ImportNode(transitions[i -
    1].Item1.GetElementsByTagName("svg")[0], true);
03089               XmlNode clonedG = Document.CreateElement("g", SVGContext.SVGNamespace);
03090               clonedG.InnerXml = clonedSVG.InnerXml;
03091               ((XmlElement)clonedG).SetAttribute("style", "visibility:  hidden");
03092               clonedSVG = clonedG;
03093
03094               {
03095                   XmlElement animate = Document.CreateElement("animate",
    SVGContext.SVGNamespace);
03096                   animate.SetAttribute("attributeName", "visibility");
03097                   animate.SetAttribute("values", "hidden;hidden;visible;visible;hidden;hidden");
03098                   animate.SetAttribute("dur",
    totalDuration.ToString(System.Globalization.CultureInfo.InvariantCulture) + "ms");
03099                   animate.SetAttribute("repeatCount", repeatCount);
03100                   animate.SetAttribute("fill", "freeze");
03101                   animate.SetAttribute("keyTimes", "0;" + Math.Min(1, currentTime /
    totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" + Math.Min(1,
    currentTime / totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" +
    Math.Min(1, (currentTime + transitions[i - 1].Item3.Duration * durationScaling) /
    totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" + Math.Min(1,
    (currentTime + transitions[i - 1].Item3.Duration * durationScaling) /
    totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";1");
03102                   clonedSVG.InsertBefore(animate, clonedSVG.FirstChild);
03103               }
03104
03105               currentElement.AppendChild(clonedSVG);
03106
03107               Dictionary<string, XmlElement> taggedStart = GetTaggedElements(clonedSVG);
03108               Dictionary<string, XmlElement> taggedEnd = GetTaggedElements(transitions[i -
    1].Item2.GetElementsByTagName("svg")[0]);
03109
03110               string overallEasing = null;
03111
03112               if (transitions[i - 1].Item3.OverallEasing is SplineEasing spline)
03113               {
03114                   overallEasing = "0.5 0.5 0.5 0.5; " +
    spline.ControlPoint1.X.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
    spline.ControlPoint1.Y.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
03115
    spline.ControlPoint2.X.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
    spline.ControlPoint2.Y.ToString(System.Globalization.CultureInfo.InvariantCulture) + "; 0.5 0.5 0.5
    0.5";
03116               }
03117
03118               foreach (KeyValuePair<string, XmlElement> kvp in taggedStart)
03119               {
```

```
03120                         if (taggedEnd.TryGetValue(kvp.Key, out XmlElement endElement))
03121                         {
03122                             List<(string, string, string)> attributesToChange = new List<(string,
      string, string)>();
03123 
03124                             foreach (XmlAttribute attr in kvp.Value.Attributes)
03125                             {
03126                                 if (endElement.HasAttribute(attr.Name))
03127                                 {
03128                                     string oldValue = attr.Value;
03129                                     string newValue = endElement.GetAttribute(attr.Name);
03130 
03131                                     if (oldValue != newValue)
03132                                     {
03133                                         attributesToChange.Add((attr.Name, oldValue, newValue));
03134                                     }
03135                                 }
03136                             }
03137 
03138                             string easing = overallEasing;
03139 
03140                             string itemId = kvp.Key.Substring(kvp.Key.IndexOf("://") + 3);
03141 
03142                             if (kvp.Value.Name == "linearGradient" || kvp.Value.Name ==
      "radialGradient")
03143                             {
03144                                 itemId = itemId.Substring(itemId.IndexOf("_") + 1);
03145                             }
03146                             else if (kvp.Value.Name == "stop")
03147                             {
03148                                 itemId = itemId.Substring(itemId.IndexOf("_") + 1);
03149                                 itemId = itemId.Substring(0, itemId.IndexOf("/"));
03150                             }
03151 
03152                             if (transitions[i - 1].Item3.Easings != null && transitions[i -
      1].Item3.Easings.TryGetValue(itemId, out IEasing currEasing) && currEasing is SplineEasing currSpline)
03153                             {
03154                                 easing = "0.5 0.5 0.5 0.5; " +
      currSpline.ControlPoint1.X.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      currSpline.ControlPoint1.Y.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
03155 
      currSpline.ControlPoint2.X.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      currSpline.ControlPoint2.Y.ToString(System.Globalization.CultureInfo.InvariantCulture) + "; 0.5 0.5
      0.5 0.5";
03156                             }
03157 
03158                             for (int j = 0; j < attributesToChange.Count; j++)
03159                             {
03160                                 kvp.Value.RemoveAttribute(attributesToChange[j].Item1);
03161 
03162                                 if (attributesToChange[j].Item1 != "transform")
03163                                 {
03164                                     XmlElement animate = Document.CreateElement("animate",
      SVGContext.SVGNamespace);
03165                                     animate.SetAttribute("attributeName",
      attributesToChange[j].Item1);
03166                                     animate.SetAttribute("values", attributesToChange[j].Item2 + ";" +
      attributesToChange[j].Item2 + ";" + attributesToChange[j].Item3 + ";" + attributesToChange[j].Item3);
03167                                     animate.SetAttribute("dur",
      totalDuration.ToString(System.Globalization.CultureInfo.InvariantCulture) + "ms");
03168                                     animate.SetAttribute("repeatCount", repeatCount);
03169                                     animate.SetAttribute("fill", "freeze");
03170                                     animate.SetAttribute("keyTimes", "0;" + Math.Min(1, currentTime /
      totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" + Math.Min(1,
      (currentTime + transitions[i - 1].Item3.Duration * durationScaling) /
      totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";1");
03171 
03172                                     if (!string.IsNullOrEmpty(easing))
03173                                     {
03174                                         animate.SetAttribute("calcMode", "spline");
03175                                         animate.SetAttribute("keySplines", easing);
03176                                     }
03177 
03178                                     if (kvp.Value.HasChildNodes)
03179                                     {
03180                                         kvp.Value.InsertBefore(animate, kvp.Value.FirstChild);
03181                                     }
03182                                     else
03183                                     {
03184                                         kvp.Value.AppendChild(animate);
03185                                     }
03186                                 }
03187                                 else
03188                                 {
03189                                     if (attributesToChange[j].Item2.StartsWith("matrix") &&
      attributesToChange[j].Item3.StartsWith("matrix"))
03190                                     {
```

```
03191                                       (Point, double, Size, Size) transformStart =
      DecomposeMatrix(attributesToChange[j].Item2);
03192                                       (Point, double, Size, Size) transformEnd =
      DecomposeMatrix(attributesToChange[j].Item3);
03193
03194                                       if (transformStart.Item4.Height != 0 ||
      transformEnd.Item4.Height != 0)
03195                                       {
03196                                           string skewStart =
      transformStart.Item4.Height.ToString(System.Globalization.CultureInfo.InvariantCulture);
03197                                           string skewEnd =
      transformEnd.Item4.Height.ToString(System.Globalization.CultureInfo.InvariantCulture);
03198
03199                                           XmlElement animate =
      Document.CreateElement("animateTransform", SVGContext.SVGNamespace);
03200                                           animate.SetAttribute("attributeName",
      attributesToChange[j].Item1);
03201                                           animate.SetAttribute("type", "skewY");
03202                                           animate.SetAttribute("values", skewStart + ";" + skewStart
      + ";" + skewEnd + ";" + skewEnd);
03203                                           animate.SetAttribute("dur",
      totalDuration.ToString(System.Globalization.CultureInfo.InvariantCulture) + "ms");
03204                                           animate.SetAttribute("repeatCount", repeatCount);
03205                                           animate.SetAttribute("fill", "freeze");
03206                                           animate.SetAttribute("additive", "sum");
03207                                           animate.SetAttribute("keyTimes", "0;" + Math.Min(1,
      currentTime / totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" +
      Math.Min(1, (currentTime + transitions[i - 1].Item3.Duration * durationScaling) /
      totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";1");
03208                                           if (!string.IsNullOrEmpty(easing))
03209                                           {
03210                                               animate.SetAttribute("calcMode", "spline");
03211                                               animate.SetAttribute("keySplines", easing);
03212                                           }
03213
03214                                           if (kvp.Value.HasChildNodes)
03215                                           {
03216                                               kvp.Value.InsertBefore(animate, kvp.Value.FirstChild);
03217                                           }
03218                                           else
03219                                           {
03220                                               kvp.Value.AppendChild(animate);
03221                                           }
03222                                       }
03223
03224                                       if (transformStart.Item4.Width != 0 ||
      transformEnd.Item4.Width != 0)
03225                                       {
03226                                           string skewStart =
      transformStart.Item4.Width.ToString(System.Globalization.CultureInfo.InvariantCulture);
03227                                           string skewEnd =
      transformEnd.Item4.Width.ToString(System.Globalization.CultureInfo.InvariantCulture);
03228
03229                                           XmlElement animate =
      Document.CreateElement("animateTransform", SVGContext.SVGNamespace);
03230                                           animate.SetAttribute("attributeName",
      attributesToChange[j].Item1);
03231                                           animate.SetAttribute("type", "skewX");
03232                                           animate.SetAttribute("values", skewStart + ";" + skewStart
      + ";" + skewEnd + ";" + skewEnd);
03233                                           animate.SetAttribute("dur",
      totalDuration.ToString(System.Globalization.CultureInfo.InvariantCulture) + "ms");
03234                                           animate.SetAttribute("repeatCount", repeatCount);
03235                                           animate.SetAttribute("fill", "freeze");
03236                                           animate.SetAttribute("additive", "sum");
03237                                           animate.SetAttribute("keyTimes", "0;" + Math.Min(1,
      currentTime / totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" +
      Math.Min(1, (currentTime + transitions[i - 1].Item3.Duration * durationScaling) /
      totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";1");
03238                                           if (!string.IsNullOrEmpty(easing))
03239                                           {
03240                                               animate.SetAttribute("calcMode", "spline");
03241                                               animate.SetAttribute("keySplines", easing);
03242                                           }
03243
03244                                           if (kvp.Value.HasChildNodes)
03245                                           {
03246                                               kvp.Value.InsertBefore(animate, kvp.Value.FirstChild);
03247                                           }
03248                                           else
03249                                           {
03250                                               kvp.Value.AppendChild(animate);
03251                                           }
03252                                       }
03253
03254                                       if (transformStart.Item3.Width != 0 ||
      transformStart.Item3.Height != 0 || transformEnd.Item3.Width != 0 || transformEnd.Item3.Height != 0)
```

```
03255                                       {
03256                                           string scaleStart =
      transformStart.Item3.Width.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      transformStart.Item3.Height.ToString(System.Globalization.CultureInfo.InvariantCulture);
03257                                           string scaleEnd =
      transformEnd.Item3.Width.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      transformEnd.Item3.Height.ToString(System.Globalization.CultureInfo.InvariantCulture);
03258
03259                                           XmlElement animate =
      Document.CreateElement("animateTransform", SVGContext.SVGNamespace);
03260                                           animate.SetAttribute("attributeName",
      attributesToChange[j].Item1);
03261                                           animate.SetAttribute("type", "scale");
03262                                           animate.SetAttribute("values", scaleStart + ";" +
      scaleStart + ";" + scaleEnd + ";" + scaleEnd);
03263                                           animate.SetAttribute("dur",
      totalDuration.ToString(System.Globalization.CultureInfo.InvariantCulture) + "ms");
03264                                           animate.SetAttribute("repeatCount", repeatCount);
03265                                           animate.SetAttribute("fill", "freeze");
03266                                           animate.SetAttribute("additive", "sum");
03267                                           animate.SetAttribute("keyTimes", "0;" + Math.Min(1,
      currentTime / totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" +
      Math.Min(1, (currentTime + transitions[i - 1].Item3.Duration * durationScaling) /
      totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";1");
03268                                           if (!string.IsNullOrEmpty(easing))
03269                                           {
03270                                               animate.SetAttribute("calcMode", "spline");
03271                                               animate.SetAttribute("keySplines", easing);
03272                                           }
03273
03274                                           if (kvp.Value.HasChildNodes)
03275                                           {
03276                                               kvp.Value.InsertBefore(animate, kvp.Value.FirstChild);
03277                                           }
03278                                           else
03279                                           {
03280                                               kvp.Value.AppendChild(animate);
03281                                           }
03282                                       }
03283
03284                                       if (transformStart.Item2 != 0 || transformEnd.Item2 != 0)
03285                                       {
03286                                           string rotateStart = (transformStart.Item2 * 180 /
      Math.PI).ToString(System.Globalization.CultureInfo.InvariantCulture);
03287                                           string rotateEnd = (transformEnd.Item2 * 180 /
      Math.PI).ToString(System.Globalization.CultureInfo.InvariantCulture);
03288
03289                                           XmlElement animate =
      Document.CreateElement("animateTransform", SVGContext.SVGNamespace);
03290                                           animate.SetAttribute("attributeName",
      attributesToChange[j].Item1);
03291                                           animate.SetAttribute("type", "rotate");
03292                                           animate.SetAttribute("values", rotateStart + ";" +
      rotateStart + ";" + rotateEnd + ";" + rotateEnd);
03293                                           animate.SetAttribute("dur",
      totalDuration.ToString(System.Globalization.CultureInfo.InvariantCulture) + "ms");
03294                                           animate.SetAttribute("repeatCount", repeatCount);
03295                                           animate.SetAttribute("fill", "freeze");
03296                                           animate.SetAttribute("additive", "sum");
03297                                           animate.SetAttribute("keyTimes", "0;" + Math.Min(1,
      currentTime / totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" +
      Math.Min(1, (currentTime + transitions[i - 1].Item3.Duration * durationScaling) /
      totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";1");
03298                                           if (!string.IsNullOrEmpty(easing))
03299                                           {
03300                                               animate.SetAttribute("calcMode", "spline");
03301                                               animate.SetAttribute("keySplines", easing);
03302                                           }
03303
03304                                           if (kvp.Value.HasChildNodes)
03305                                           {
03306                                               kvp.Value.InsertBefore(animate, kvp.Value.FirstChild);
03307                                           }
03308                                           else
03309                                           {
03310                                               kvp.Value.AppendChild(animate);
03311                                           }
03312                                       }
03313
03314                                       if (transformStart.Item1.X != 0 || transformStart.Item1.Y != 0
      || transformEnd.Item1.X != 0 || transformEnd.Item1.Y != 0)
03315                                       {
03316                                           string translateStart =
      transformStart.Item1.X.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
      transformStart.Item1.Y.ToString(System.Globalization.CultureInfo.InvariantCulture);
03317                                           string translateEnd =
      transformEnd.Item1.X.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
```

```
transformEnd.Item1.Y.ToString(System.Globalization.CultureInfo.InvariantCulture);
03318
03319                                           XmlElement animate =
Document.CreateElement("animateTransform", SVGContext.SVGNamespace);
03320                                           animate.SetAttribute("attributeName",
attributesToChange[j].Item1);
03321                                           animate.SetAttribute("type", "translate");
03322                                           animate.SetAttribute("values", translateStart + ";" +
translateStart + ";" + translateEnd + ";" + translateEnd);
03323                                           animate.SetAttribute("dur",
totalDuration.ToString(System.Globalization.CultureInfo.InvariantCulture) + "ms");
03324                                           animate.SetAttribute("repeatCount", repeatCount);
03325                                           animate.SetAttribute("fill", "freeze");
03326                                           animate.SetAttribute("additive", "sum");
03327                                           animate.SetAttribute("keyTimes", "0;" + Math.Min(1,
currentTime / totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" +
Math.Min(1, (currentTime + transitions[i - 1].Item3.Duration * durationScaling) /
totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";1");
03328                                           if (!string.IsNullOrEmpty(easing))
03329                                           {
03330                                               animate.SetAttribute("calcMode", "spline");
03331                                               animate.SetAttribute("keySplines", easing);
03332                                           }
03333
03334                                           if (kvp.Value.HasChildNodes)
03335                                           {
03336                                               kvp.Value.InsertBefore(animate, kvp.Value.FirstChild);
03337                                           }
03338                                           else
03339                                           {
03340                                               kvp.Value.AppendChild(animate);
03341                                           }
03342                                       }
03343                                   }
03344                                   else
03345                                   {
03346                                       throw new NotImplementedException("Invalid transform!");
03347                                   }
03348                               }
03349                           }
03350                       }
03351                   }
03352
03353                   currentTime += transitions[i - 1].Item3.Duration * durationScaling;
03354               }
03355
03356               {
03357                   XmlNode clonedSVG =
Document.ImportNode(frames[i].Item1.GetElementsByTagName("svg")[0], true);
03358
03359                   string prevStyle = ((XmlElement)clonedSVG).GetAttribute("style");
03360
03361                   if (i == 0)
03362                   {
03363                       if (!string.IsNullOrEmpty(prevStyle))
03364                       {
03365                           ((XmlElement)clonedSVG).SetAttribute("style", prevStyle + " visibility:
visible;");
03366                       }
03367                       else
03368                       {
03369                           ((XmlElement)clonedSVG).SetAttribute("style", "visibility:  visible;");
03370                       }
03371                   }
03372                   else
03373                   {
03374                       if (!string.IsNullOrEmpty(prevStyle))
03375                       {
03376                           ((XmlElement)clonedSVG).SetAttribute("style", prevStyle + " visibility:
hidden;");
03377                       }
03378                       else
03379                       {
03380                           ((XmlElement)clonedSVG).SetAttribute("style", "visibility:  hidden;");
03381                       }
03382                   }
03383
03384                   XmlElement animate = Document.CreateElement("animate", SVGContext.SVGNamespace);
03385                   animate.SetAttribute("attributeName", "visibility");
03386                   animate.SetAttribute("values", "hidden;hidden;visible;visible;hidden;hidden");
03387                   animate.SetAttribute("dur",
totalDuration.ToString(System.Globalization.CultureInfo.InvariantCulture) + "ms");
03388                   animate.SetAttribute("repeatCount", repeatCount);
03389
03390                   if (i < frames.Length - 1)
03391                   {
03392                       animate.SetAttribute("fill", "freeze");
```

```
03393                     }
03394
03395                     animate.SetAttribute("keyTimes", "0;" + Math.Min(1, currentTime /
      totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" + Math.Min(1,
      currentTime / totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" +
      Math.Min(1, (currentTime + durations[i]) /
      totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" + Math.Min(1,
      (currentTime + durations[i]) /
      totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";1");
03396                     clonedSVG.InsertBefore(animate, clonedSVG.FirstChild);
03397
03398                     currentElement.AppendChild(clonedSVG);
03399                 }
03400
03401                 currentTime += durations[i];
03402             }
03403
03404             if (includeControls)
03405             {
03406                 double controlsHeight = animation.Height * 0.08;
03407
03408                 double controlsWidth = animation.Width * 0.8;
03409
03410                 if (controlsWidth < controlsHeight * 5)
03411                 {
03412                     controlsHeight = controlsWidth / 5;
03413                 }
03414
03415                 currentElement.AppendChild(Document.ImportNode(GetAnimationControls(animation.Width,
      animation.Height, controlsWidth, controlsHeight, animation.Duration * durationScaling,
      animation.RepeatCount).GetElementsByTagName("svg")[0], true));
03416             }
03417
03418             return Document;
03419         }
03420
03421         private static XmlDocument GetAnimationControls(double width, double height, double
      controlsWidth, double controlsHeight, double totalDuration, int repeatCount)
03422         {
03423             double controlsMargin = controlsHeight / 12;
03424             double controlRadius = controlsHeight / 6;
03425
03426             Page pag = new Page(width, height);
03427
03428             Graphics gpr = pag.Graphics;
03429
03430             GraphicsPath background = new GraphicsPath().MoveTo(width * 0.5 - controlsWidth * 0.5 +
      controlRadius, height - controlsMargin - controlsHeight).LineTo(width * 0.5 + controlsWidth * 0.5 -
      controlRadius, height - controlsMargin - controlsHeight);
03431             background.Arc(width * 0.5 + controlsWidth * 0.5 - controlRadius, height - controlsMargin
      - controlsHeight + controlRadius, controlRadius, -Math.PI / 2, 0).LineTo(width * 0.5 + controlsWidth *
      0.5, height - controlsMargin - controlRadius);
03432             background.Arc(width * 0.5 + controlsWidth * 0.5 - controlRadius, height - controlsMargin
      - controlRadius, controlRadius, 0, Math.PI / 2).LineTo(width * 0.5 - controlsWidth * 0.5 +
      controlRadius, height - controlsMargin);
03433             background.Arc(width * 0.5 - controlsWidth * 0.5 + controlRadius, height - controlsMargin
      - controlRadius, controlRadius, Math.PI / 2, Math.PI).LineTo(width * 0.5 - controlsWidth * 0.5, height
      - controlsMargin - controlsHeight + controlRadius);
03434             background.Arc(width * 0.5 - controlsWidth * 0.5 + controlRadius, height - controlsMargin
      - controlsHeight + controlRadius, controlRadius, Math.PI, 3 * Math.PI / 2).Close();
03435
03436             gpr.FillPath(background, Colour.FromRgba(0, 0, 0, 0.5), tag:  "timeLineBackground");
03437
03438             double roundedCornerSize = 0.25;
03439
03440             LinearGradientBrush buttonBrush = new LinearGradientBrush(new Point(0, controlsHeight),
      new Point(0, 0), new GradientStop(Colours.White, 1), new GradientStop(Colour.FromRgb(220, 220, 220),
      0));
03441
03442             gpr.Save();
03443             gpr.Translate(width * 0.5 - controlsWidth * 0.5 + controlsHeight * 0.5, height -
      controlsHeight - controlsMargin);
03444             GraphicsPath playButton = new GraphicsPath().MoveTo(0, controlsHeight * 0.2 +
      controlsHeight * 0.6 * roundedCornerSize).LineTo(0, controlsHeight * 0.2 + controlsHeight * 0.6 * (1 -
      roundedCornerSize)).CubicBezierTo(0, controlsHeight * 0.8, 0, controlsHeight * 0.8, controlsHeight *
      0.5 * roundedCornerSize, controlsHeight * 0.5 + controlsHeight * 0.3 * (1 -
      roundedCornerSize)).LineTo(controlsHeight * 0.5 * (1 - roundedCornerSize), controlsHeight * 0.5 +
      controlsHeight * 0.3 * roundedCornerSize);
03445             playButton.CubicBezierTo(controlsHeight * 0.5, controlsHeight * 0.5, controlsHeight * 0.5,
      controlsHeight * 0.5, controlsHeight * 0.5 * (1 - roundedCornerSize), controlsHeight * 0.5 -
      controlsHeight * 0.3 * roundedCornerSize).LineTo(controlsHeight * 0.5 * roundedCornerSize,
      controlsHeight * 0.5 - controlsHeight * 0.3 * (1 - roundedCornerSize));
03446             playButton.CubicBezierTo(0, controlsHeight * 0.2, 0, controlsHeight * 0.2, 0,
      controlsHeight * 0.2 + controlsHeight * 0.6 * roundedCornerSize).Close();
03447             gpr.FillPath(playButton, buttonBrush, "playButton");
03448
03449             GraphicsPath pauseButton = new GraphicsPath().MoveTo(controlsHeight * 0.2 *
```

```
roundedCornerSize, controlsHeight * 0.25).LineTo(controlsHeight * 0.2 * (1 - roundedCornerSize),
controlsHeight * 0.25).CubicBezierTo(controlsHeight * 0.2, controlsHeight * 0.25, controlsHeight *
0.2, controlsHeight * 0.25, controlsHeight * 0.2, controlsHeight * 0.25 + controlsHeight * 0.5 *
roundedCornerSize);
03450           pauseButton.LineTo(controlsHeight * 0.2, controlsHeight * 0.25 + controlsHeight * 0.5 * (1
- roundedCornerSize)).CubicBezierTo(controlsHeight * 0.2, controlsHeight * 0.75, controlsHeight * 0.2,
controlsHeight * 0.75, controlsHeight * 0.2 * (1 - roundedCornerSize), controlsHeight *
0.75).LineTo(controlsHeight * 0.2 * roundedCornerSize, controlsHeight * 0.75);
03451           pauseButton.CubicBezierTo(0, controlsHeight * 0.75, 0, controlsHeight * 0.75, 0,
controlsHeight * 0.25 + controlsHeight * 0.5 * (1 - roundedCornerSize)).LineTo(0, controlsHeight *
0.25 + controlsHeight * 0.5 * roundedCornerSize).CubicBezierTo(0, controlsHeight * 0.25, 0,
controlsHeight * 0.25, controlsHeight * 0.2 * roundedCornerSize, controlsHeight * 0.25).Close();
03452           pauseButton.MoveTo(controlsHeight * 0.3 + controlsHeight * 0.2 * roundedCornerSize,
controlsHeight * 0.25).LineTo(controlsHeight * 0.3 + controlsHeight * 0.2 * (1 - roundedCornerSize),
controlsHeight * 0.25).CubicBezierTo(controlsHeight * 0.3 + controlsHeight * 0.2, controlsHeight *
0.25, controlsHeight * 0.3 + controlsHeight * 0.2, controlsHeight * 0.25, controlsHeight * 0.3 +
controlsHeight * 0.2, controlsHeight * 0.25 + controlsHeight * 0.5 * roundedCornerSize);
03453           pauseButton.LineTo(controlsHeight * 0.3 + controlsHeight * 0.2, controlsHeight * 0.25 +
controlsHeight * 0.5 * (1 - roundedCornerSize)).CubicBezierTo(controlsHeight * 0.3 + controlsHeight *
0.2, controlsHeight * 0.75, controlsHeight * 0.3 + controlsHeight * 0.2, controlsHeight * 0.75,
controlsHeight * 0.3 + controlsHeight * 0.2 * (1 - roundedCornerSize), controlsHeight *
0.75).LineTo(controlsHeight * 0.3 + controlsHeight * 0.2 * roundedCornerSize, controlsHeight * 0.75);
03454           pauseButton.CubicBezierTo(controlsHeight * 0.3, controlsHeight * 0.75, controlsHeight *
0.3, controlsHeight * 0.75, controlsHeight * 0.3, controlsHeight * 0.25 + controlsHeight * 0.5 * (1 -
roundedCornerSize)).LineTo(controlsHeight * 0.3, controlsHeight * 0.25 + controlsHeight * 0.5 *
roundedCornerSize).CubicBezierTo(controlsHeight * 0.3, controlsHeight * 0.25, controlsHeight * 0.3,
controlsHeight * 0.25, controlsHeight * 0.3 + controlsHeight * 0.2 * roundedCornerSize, controlsHeight
* 0.25).Close();
03455           gpr.FillPath(pauseButton, buttonBrush, "pauseButton");
03456
03457           gpr.FillRectangle(0, controlsHeight * 0.2, controlsHeight * 0.5, controlsHeight * 0.6,
Colour.FromRgba(0, 0, 0, 0), tag:  "playButtonHitbox");
03458           gpr.FillRectangle(0, controlsHeight * 0.2, controlsHeight * 0.5, controlsHeight * 0.6,
Colour.FromRgba(0, 0, 0, 0), tag:  "pauseButtonHitbox");
03459
03460           double timeLineStartX = controlsHeight * 0.5 + controlsHeight * 0.5 + controlsHeight / 3;
03461
03462           gpr.Translate(timeLineStartX, 0);
03463
03464           GraphicsPath timeLine = new GraphicsPath().MoveTo(controlsHeight * 0.1, controlsHeight *
0.4).LineTo(controlsWidth - timeLineStartX - controlsHeight - controlsHeight * 0.1 - controlsHeight /
3, controlsHeight * 0.4);
03465           timeLine.Arc(controlsWidth - timeLineStartX - controlsHeight - controlsHeight * 0.1 -
controlsHeight / 3, controlsHeight * 0.5, controlsHeight * 0.1, -Math.PI / 2, Math.PI /
2).LineTo(controlsHeight * 0.1, controlsHeight * 0.6);
03466           timeLine.Arc(controlsHeight * 0.1, controlsHeight * 0.5, controlsHeight * 0.1, Math.PI /
2, 3 * Math.PI / 2).Close();
03467
03468           gpr.FillPath(timeLine, new LinearGradientBrush(new Point(0, controlsHeight * 0.6), new
Point(0, controlsHeight * 0.4), new GradientStop(Colour.FromRgba(255, 255, 255, 80), 0), new
GradientStop(Colour.FromRgba(255, 255, 255, 40), 1)), tag:  "timeLine");
03469           gpr.StrokePath(timeLine, Colour.FromRgba(255, 255, 255, 180), tag:  "timeLineStroke");
03470
03471           gpr.Restore();
03472
03473           GraphicsPath thumbPath = new GraphicsPath().MoveTo(-controlsHeight * 0.2, controlsHeight *
0.3).LineTo(controlsHeight * 0.2, controlsHeight * 0.3);
03474           thumbPath.Arc(controlsHeight * 0.2, controlsHeight * 0.5, controlsHeight * 0.2, -Math.PI /
2, Math.PI / 2).LineTo(-controlsHeight * 0.2, controlsHeight * 0.7);
03475           thumbPath.Arc(-controlsHeight * 0.2, controlsHeight * 0.5, controlsHeight * 0.2, Math.PI /
2, 3 * Math.PI / 2).Close();
03476
03477           Graphics thumbStart = new Graphics();
03478           thumbStart.Translate(width * 0.5 - controlsWidth * 0.5 + controlsHeight * 0.5 +
timeLineStartX, height - controlsHeight - controlsMargin);
03479           thumbStart.FillPath(thumbPath, buttonBrush, tag:  "thumb");
03480
03481           Graphics thumbEnd = new Graphics();
03482           thumbEnd.Translate(width * 0.5 + controlsWidth * 0.5 - controlsHeight * 0.5 -
controlsHeight * 0.4, height - controlsHeight - controlsMargin);
03483           thumbEnd.FillPath(thumbPath, buttonBrush, tag:  "thumb");
03484
03485           Animation thumbAnimation = new Animation(width, height, 1) { RepeatCount = repeatCount };
03486           thumbAnimation.AddFrame(new Frame(thumbStart, 0));
03487           thumbAnimation.AddFrame(new Frame(thumbEnd, 0), new Transition(totalDuration));
03488
03489           XmlDocument animationDoc = thumbAnimation.SaveAsAnimatedSVG();
03490
03491           XmlNodeList svgNodes = animationDoc.GetElementsByTagName("svg");
03492
03493           for (int i = 1; i < svgNodes.Count; i++)
03494           {
03495
((XmlElement)svgNodes[i]).RemoveChild(((XmlElement)svgNodes[i]).GetElementsByTagName("path")[0]);
03496           }
03497
03498           ((XmlElement)animationDoc.GetElementsByTagName("svg")[0]).SetAttribute("style",
```

```
      "pointer-events:  none");
03499
03500             XmlDocument doc = pag.SaveAsSVG();
03501
03502             XmlNodeList paths = doc.GetElementsByTagName("path");
03503
03504             for (int i = 0; i < paths.Count; i++)
03505             {
03506                 if (((XmlElement)paths[i]).GetAttribute("id") == "timeLine")
03507                 {
03508                     ((XmlElement)paths[i]).SetAttribute("style", "cursor:  pointer;");
03509                 }
03510
03511                 if (((XmlElement)paths[i]).GetAttribute("id") == "playButtonHitbox" ||
      ((XmlElement)paths[i]).GetAttribute("id") == "pauseButtonHitbox")
03512                 {
03513                     ((XmlElement)paths[i]).SetAttribute("style", "cursor:  pointer");
03514                 }
03515             }
03516
03517             XmlNode animationNode = doc.ImportNode(animationDoc.GetElementsByTagName("svg")[0], true);
03518             doc.GetElementsByTagName("svg")[0].AppendChild(animationNode);
03519             ((XmlElement)doc.GetElementsByTagName("svg")[0]).SetAttribute("id", "animationControls");
03520
03521             ((XmlElement)doc.GetElementsByTagName("svg")[0]).SetAttribute("style", "font-synthesis:
      none; transition:  opacity 500ms;");
03522
03523             using (Stream sr =
      Assembly.GetExecutingAssembly().GetManifestResourceStream("VectSharp.SVG.AnimationControls.js"))
03524             using (StreamReader reader = new StreamReader(sr))
03525             {
03526                 string javascript = reader.ReadToEnd();
03527                 javascript = javascript.Replace("@@totalLength@@", (totalDuration /
      1000).ToString(System.Globalization.CultureInfo.InvariantCulture));
03528
03529                 XmlElement script = doc.CreateElement("script", SVGContext.SVGNamespace);
03530                 script.SetAttribute("type", "text/javascript");
03531                 script.InnerText = javascript;
03532
03533                 doc.GetElementsByTagName("svg")[0].AppendChild(script);
03534             }
03535
03536             return doc;
03537         }
03538
03539 /// <summary>
03540 /// Render the animation to an SVG stream, using SVG animations.
03541 /// </summary>
03542 /// <param name="animation">The <see cref="Animation"/> to render.</param>
03543 /// <param name="stream">The <see cref="Stream"/> on which the SVG document will be written.</param>
03544 /// <param name="includeControls">If this is <see langword="true"/>, the generated SVG file will
      contain playback controls that use Javascript to play/pause the animation and change the current
      time.</param>
03545 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the
      animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note
      that this does not affect the frame rate of the animation.</param>
03546 /// <param name="textOption">Defines whether the used fonts should be included in the file.</param>
03547 /// <param name="linkDestinations">A dictionary associating element tags to link targets.  If this is
      provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to
      the destination specified in the dictionary.  If the destination starts with a hash (#), it is
      interpreted as the tag of another object in the current document; otherwise, it is interpreted as an
      external URI.</param>
03548 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
      rendering the image.</param>
03549         public static void SaveAsAnimatedSVG(this Animation animation, Stream stream, bool
      includeControls = false, double durationScaling = 1, TextOptions textOption = TextOptions.SubsetFonts,
      Dictionary<string, string> linkDestinations = null, FilterOption filterOption = default)
03550         {
03551             XmlDocument document = SaveAsAnimatedSVG(animation, includeControls, durationScaling,
      textOption, linkDestinations, filterOption);
03552             WriteXMLToStream(document.DocumentElement, stream);
03553         }
03554
03555 /// <summary>
03556 /// Render the animation to an SVG file, using SVG animations.
03557 /// </summary>
03558 /// <param name="animation">The <see cref="Animation"/> to render.</param>
03559 /// <param name="fileName">The output file that will be created.</param>
03560 /// <param name="includeControls">If this is <see langword="true"/>, the generated SVG file will
      contain playback controls that use Javascript to play/pause the animation and change the current
      time.</param>
03561 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the
      animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note
      that this does not affect the frame rate of the animation.</param>
03562 /// <param name="textOption">Defines whether the used fonts should be included in the file.</param>
03563 /// <param name="linkDestinations">A dictionary associating element tags to link targets.  If this is
      provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to
```

```
the destination specified in the dictionary.  If the destination starts with a hash (#), it is
interpreted as the tag of another object in the current document; otherwise, it is interpreted as an
external URI.</param>
03564 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
rendering the image.</param>
03565         public static void SaveAsAnimatedSVG(this Animation animation, string fileName, bool
includeControls = false, double durationScaling = 1, TextOptions textOption = TextOptions.SubsetFonts,
Dictionary<string, string> linkDestinations = null, FilterOption filterOption = default)
03566         {
03567             using (FileStream fs = File.Create(fileName))
03568             {
03569                 SaveAsAnimatedSVG(animation, fs, includeControls, durationScaling, textOption,
linkDestinations, filterOption);
03570             }
03571         }
03572
03573 /// <summary>
03574 /// Render the animation to an SVG document, encoding discrete frames.
03575 /// </summary>
03576 /// <param name="animation">The <see cref="Animation"/> to render.</param>
03577 /// <param name="includeControls">If this is <see langword="true"/>, the generated SVG file will
contain playback controls that use Javascript to play/pause the animation and change the current
time.</param>
03578 /// <param name="frameRate">The target frame rate of the animation, in frames-per-second
(fps).</param>
03579 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the
animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note
that this does not affect the frame rate of the animation.</param>
03580 /// <param name="textOption">Defines whether the used fonts should be included in the file.</param>
03581 /// <param name="linkDestinations">A dictionary associating element tags to link targets.  If this is
provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to
the destination specified in the dictionary.  If the destination starts with a hash (#), it is
interpreted as the tag of another object in the current document; otherwise, it is interpreted as an
external URI.</param>
03582 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
rendering the image.</param>
03583 /// <returns>An <see cref="XmlDocument"/> containing the animated SVG image.</returns>
03584         public static XmlDocument SaveAsAnimatedSVGWithFrames(this Animation animation, bool
includeControls = false, double frameRate = 60, double durationScaling = 1, TextOptions textOption =
TextOptions.SubsetFonts, Dictionary<string, string> linkDestinations = null, FilterOption filterOption
= default)
03585         {
03586             int frameCount = (int)Math.Ceiling(animation.Duration * frameRate * durationScaling /
1000);
03587
03588             (XmlDocument, double)[] frames = new (XmlDocument, double)[frameCount];
03589             Dictionary<string, RasterImage> usedRasterImages = new Dictionary<string, RasterImage>();
03590             Dictionary<string, (string, FontFamily, HashSet<char>)> usedFonts = new Dictionary<string,
(string, FontFamily, HashSet<char>)>();
03591             object lockObject = new object();
03592
03593             Parallel.For(0, frameCount, i =>
03594             {
03595                 double frameTime = i / frameRate / durationScaling * 1000;
03596
03597                 double frameDuration = Math.Min((animation.Duration - frameTime) * durationScaling,
1000 / frameRate);
03598
03599                 Page pag = animation.GetFrameAtAbsolute(frameTime);
03600
03601                 frames[i] = (pag.CreateSVGDocument("frame" + i.ToString() + "://", textOption,
linkDestinations, filterOption, false, true, usedRasterImages, usedFonts, false, lockObject),
frameDuration);
03602             });
03603
03604             string repeatCount = animation.RepeatCount <= 0 ?  "indefinite" :
animation.RepeatCount.ToString();
03605
03606             double totalDuration = animation.Duration * durationScaling;
03607
03608             XmlDocument Document = new XmlDocument();
03609
03610             Document.InsertBefore(Document.CreateXmlDeclaration("1.0", "UTF-8", null),
Document.DocumentElement);
03611
03612             XmlElement currentElement = Document.CreateElement(null, "svg", SVGContext.SVGNamespace);
03613             currentElement.SetAttribute("xmlns:xlink", "http://www.w3.org/1999/xlink");
03614             currentElement.SetAttribute("viewBox", "0 0 " +
animation.Width.ToString(System.Globalization.CultureInfo.InvariantCulture) + " " +
animation.Height.ToString(System.Globalization.CultureInfo.InvariantCulture));
03615             currentElement.SetAttribute("version", "1.1");
03616             Document.AppendChild(currentElement);
03617
03618             if (usedFonts.Count > 0 && (textOption == TextOptions.SubsetFonts || textOption ==
TextOptions.EmbedFonts))
03619             {
03620                 StringBuilder cssFonts = new StringBuilder();
```

```
03621
03622                 foreach (KeyValuePair<string, (string, FontFamily, HashSet<char>)> kvp in usedFonts)
03623                 {
03624                     TrueTypeFile subsettedFont;
03625
03626                     if (textOption == TextOptions.SubsetFonts)
03627                     {
03628                         subsettedFont = kvp.Value.Item2.TrueTypeFile.SubsetFont(new
      string(kvp.Value.Item3.ToArray()));
03629                     }
03630                     else
03631                     {
03632                         subsettedFont = kvp.Value.Item2.TrueTypeFile;
03633                     }
03634
03635                     byte[] fontBytes;
03636
03637                     using (MemoryStream fontStream = new
      MemoryStream((int)subsettedFont.FontStream.Length))
03638                     {
03639                         subsettedFont.FontStream.Seek(0, SeekOrigin.Begin);
03640                         subsettedFont.FontStream.CopyTo(fontStream);
03641
03642                         fontBytes = fontStream.ToArray();
03643                     }
03644
03645
03646                     cssFonts.Append("\n    @font-face\n    {\n      font-family:  \"" +
      kvp.Value.Item1 + "\";\n      src:  url(\"data:font/ttf;charset=utf-8base64,");
03647                     cssFonts.Append(Convert.ToBase64String(fontBytes));
03648                     cssFonts.Append("\");\n    }\n  ");
03649                 }
03650
03651                 XmlElement style = Document.CreateElement("style", SVGContext.SVGNamespace);
03652                 style.InnerText = cssFonts.ToString();
03653                 currentElement.AppendChild(style);
03654             }
03655
03656             if (usedFonts.Count > 0 && textOption == TextOptions.ConvertIntoPathsUsingGlyphs)
03657             {
03658                 foreach (KeyValuePair<string, (string, FontFamily, HashSet<char>)> kvp in usedFonts)
03659                 {
03660                     string guid = Guid.NewGuid().ToString();
03661
03662                     XmlElement defs = Document.CreateElement("defs", SVGContext.SVGNamespace);
03663                     defs.SetAttribute("id", kvp.Value.Item2.FamilyName + "-glyphs-" + guid);
03664
03665                     foreach (char c in kvp.Value.Item3)
03666                     {
03667                         XmlElement charPath = Document.CreateElement("path", SVGContext.SVGNamespace);
03668                         charPath.SetAttribute("id", kvp.Key + "-" + c);
03669
03670                         TrueTypeFile.TrueTypePoint[][] glyphPaths =
      kvp.Value.Item2.TrueTypeFile.GetGlyphPath(c, kvp.Value.Item2.TrueTypeFile.GetUnitsPerEm());
03671
03672                         StringBuilder data = new StringBuilder();
03673
03674                         for (int j = 0; j < glyphPaths.Length; j++)
03675                         {
03676                             for (int k = 0; k < glyphPaths[j].Length; k++)
03677                             {
03678                                 if (k == 0)
03679                                 {
03680                                     data.Append("M " +
      glyphPaths[j][k].X.ToString(System.Globalization.CultureInfo.InvariantCulture) + "," +
      (-glyphPaths[j][k].Y).ToString(System.Globalization.CultureInfo.InvariantCulture) + " ");
03681                                 }
03682                                 else
03683                                 {
03684                                     if (glyphPaths[j][k].IsOnCurve)
03685                                     {
03686                                         data.Append("L " +
      glyphPaths[j][k].X.ToString(System.Globalization.CultureInfo.InvariantCulture) + "," +
      (-glyphPaths[j][k].Y).ToString(System.Globalization.CultureInfo.InvariantCulture) + " ");
03687                                     }
03688                                     else
03689                                     {
03690                                         data.Append("Q " +
      glyphPaths[j][k].X.ToString(System.Globalization.CultureInfo.InvariantCulture) + "," +
      (-glyphPaths[j][k].Y).ToString(System.Globalization.CultureInfo.InvariantCulture) + " ");
03691                                         data.Append(glyphPaths[j][k +
      1].X.ToString(System.Globalization.CultureInfo.InvariantCulture) + "," + (-glyphPaths[j][k +
      1].Y).ToString(System.Globalization.CultureInfo.InvariantCulture) + " ");
03692
03693                                         k++;
03694                                     }
03695                                 }
```

```
03696                               }
03697
03698                               data.Append("Z");
03699                           }
03700
03701                           charPath.SetAttribute("d", data.ToString());
03702
03703                           defs.AppendChild(charPath);
03704                       }
03705
03706                       currentElement.AppendChild(defs);
03707                   }
03708               }
03709
03710               if (usedRasterImages.Count > 0)
03711               {
03712                   XmlElement definitions = Document.CreateElement("defs", SVGContext.SVGNamespace);
03713                   currentElement.AppendChild(definitions);
03714
03715                   foreach (KeyValuePair<string, RasterImage> kvp in usedRasterImages)
03716                   {
03717                       EmbedRasterImage(Document, kvp.Key, kvp.Value, definitions);
03718                   }
03719               }
03720
03721               double currentTime = 0;
03722
03723               for (int i = 0; i < frames.Length; i++)
03724               {
03725                   XmlNode clonedSVG =
      Document.ImportNode(frames[i].Item1.GetElementsByTagName("svg")[0], true);
03726                   XmlNode clonedG = Document.CreateElement("g", SVGContext.SVGNamespace);
03727                   clonedG.InnerXml = clonedSVG.InnerXml;
03728                   if (i == 0)
03729                   {
03730                       ((XmlElement)clonedG).SetAttribute("style", "visibility:  visible");
03731                   }
03732                   else
03733                   {
03734                       ((XmlElement)clonedG).SetAttribute("style", "visibility:  hidden");
03735                   }
03736
03737                   XmlElement animate = Document.CreateElement("animate", SVGContext.SVGNamespace);
03738                   animate.SetAttribute("attributeName", "visibility");
03739                   animate.SetAttribute("values", "hidden;hidden;visible;visible;hidden;hidden");
03740                   animate.SetAttribute("dur",
      totalDuration.ToString(System.Globalization.CultureInfo.InvariantCulture) + "ms");
03741                   animate.SetAttribute("repeatCount", repeatCount);
03742                   animate.SetAttribute("keyTimes", "0;" + Math.Min(1, currentTime /
      totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" + Math.Min(1,
      currentTime / totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" +
      Math.Min(1, (currentTime + frames[i].Item2) /
      totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";" + Math.Min(1,
      (currentTime + frames[i].Item2) /
      totalDuration).ToString(System.Globalization.CultureInfo.InvariantCulture) + ";1");
03743                   clonedG.InsertBefore(animate, clonedG.FirstChild);
03744
03745                   currentElement.AppendChild(clonedG);
03746
03747                   currentTime += frames[i].Item2;
03748               }
03749
03750               if (includeControls)
03751               {
03752                   double controlsHeight = animation.Height * 0.08;
03753
03754                   double controlsWidth = animation.Width * 0.8;
03755
03756                   if (controlsWidth < controlsHeight * 5)
03757                   {
03758                       controlsHeight = controlsWidth / 5;
03759                   }
03760
03761                   currentElement.AppendChild(Document.ImportNode(GetAnimationControls(animation.Width,
      animation.Height, controlsWidth, controlsHeight, animation.Duration * durationScaling,
      animation.RepeatCount).GetElementsByTagName("svg")[0], true));
03762               }
03763
03764               return Document;
03765           }
03766
03767 /// <summary>
03768 /// Render the animation to an SVG stream, encoding discrete frames.
03769 /// </summary>
03770 /// <param name="animation">The <see cref="Animation"/> to render.</param>
03771 /// <param name="includeControls">If this is <see langword="true"/>, the generated SVG file will
      contain playback controls that use Javascript to play/pause the animation and change the current
```

```
time.</param>
03772 /// <param name="stream">The <see cref="Stream"/> on which the SVG document will be written.</param>
03773 /// <param name="frameRate">The target frame rate of the animation, in frames-per-second
(fps).</param>
03774 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the
animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note
that this does not affect the frame rate of the animation.</param>
03775 /// <param name="textOption">Defines whether the used fonts should be included in the file.</param>
03776 /// <param name="linkDestinations">A dictionary associating element tags to link targets.  If this is
provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to
the destination specified in the dictionary.  If the destination starts with a hash (#), it is
interpreted as the tag of another object in the current document; otherwise, it is interpreted as an
external URI.</param>
03777 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
rendering the image.</param>
03778         public static void SaveAsAnimatedSVGWithFrames(this Animation animation, Stream stream, bool
includeControls = false, double frameRate = 60, double durationScaling = 1, TextOptions textOption =
TextOptions.SubsetFonts, Dictionary<string, string> linkDestinations = null, FilterOption filterOption
= default)
03779         {
03780             XmlDocument document = SaveAsAnimatedSVGWithFrames(animation, includeControls, frameRate,
durationScaling, textOption, linkDestinations, filterOption);
03781             WriteXMLToStream(document.DocumentElement, stream);
03782         }
03783
03784 /// <summary>
03785 /// Render the animation to an SVG file, encoding discrete frames.
03786 /// </summary>
03787 /// <param name="animation">The <see cref="Animation"/> to render.</param>
03788 /// <param name="includeControls">If this is <see langword="true"/>, the generated SVG file will
contain playback controls that use Javascript to play/pause the animation and change the current
time.</param>
03789 /// <param name="fileName">The output file that will be created.</param>
03790 /// <param name="frameRate">The target frame rate of the animation, in frames-per-second
(fps).</param>
03791 /// <param name="durationScaling">A scaling factor that will be applied to all durations in the
animation.  Values greater than 1 slow down the animation, values smaller than 1 accelerate it.  Note
that this does not affect the frame rate of the animation.</param>
03792 /// <param name="textOption">Defines whether the used fonts should be included in the file.</param>
03793 /// <param name="linkDestinations">A dictionary associating element tags to link targets.  If this is
provided, objects that have been drawn with a tag contained in the dictionary will become hyperlink to
the destination specified in the dictionary.  If the destination starts with a hash (#), it is
interpreted as the tag of another object in the current document; otherwise, it is interpreted as an
external URI.</param>
03794 /// <param name="filterOption">Defines how and whether image filters should be rasterised when
rendering the image.</param>
03795         public static void SaveAsAnimatedSVGWithFrames(this Animation animation, string fileName, bool
includeControls = false, double frameRate = 60, double durationScaling = 1, TextOptions textOption =
TextOptions.SubsetFonts, Dictionary<string, string> linkDestinations = null, FilterOption filterOption
= default)
03796         {
03797             using (FileStream fs = File.Create(fileName))
03798             {
03799                 SaveAsAnimatedSVGWithFrames(animation, fs, includeControls, frameRate,
durationScaling, textOption, linkDestinations, filterOption);
03800             }
03801         }
03802
03803         private static readonly Regex SimpleMatrixTransform = new
Regex("\\s*matrix\\([^\\)]+\\)\\s*$", RegexOptions.Compiled);
03804         private static readonly Regex MatrixAndScaleTransform = new
Regex("\\s*matrix\\([^\\)]+\\)\\s*,\\s*scale\\([^\\)]+\\)\\s*$", RegexOptions.Compiled);
03805
03806
03807         // Adapted from https://math.stackexchange.com/a/2888105
03808         private static (Point, double, Size, Size) DecomposeMatrix(string transformMatrix)
03809         {
03810             double a, b, c, d, e, f;
03811
03812             if (SimpleMatrixTransform.IsMatch(transformMatrix))
03813             {
03814                 transformMatrix = transformMatrix.Trim().Substring(7);
03815                 transformMatrix = transformMatrix.Substring(0, transformMatrix.Length - 1);
03816
03817                 double[] matrixElements = (from el in transformMatrix.Split(',') select
double.Parse(el, System.Globalization.CultureInfo.InvariantCulture)).ToArray();
03818
03819                 a = matrixElements[0];
03820                 b = matrixElements[1];
03821                 c = matrixElements[2];
03822                 d = matrixElements[3];
03823                 e = matrixElements[4];
03824                 f = matrixElements[5];
03825             }
03826             else if (MatrixAndScaleTransform.IsMatch(transformMatrix))
03827             {
03828                 string fullMatrix = transformMatrix;
```

```
03829
03830                transformMatrix = transformMatrix.Trim().Substring(7);
03831                transformMatrix = transformMatrix.Substring(0, transformMatrix.IndexOf(")"));
03832
03833                double[] matrixElements = (from el in transformMatrix.Split(',') select
      double.Parse(el, System.Globalization.CultureInfo.InvariantCulture)).ToArray();
03834
03835                a = matrixElements[0];
03836                b = matrixElements[1];
03837                c = matrixElements[2];
03838                d = matrixElements[3];
03839                e = matrixElements[4];
03840                f = matrixElements[5];
03841
03842                string scaleString = fullMatrix.Substring(fullMatrix.IndexOf("scale(") + 6).Trim();
03843                scaleString = scaleString.Substring(0, scaleString.Length - 1);
03844
03845                double[] scaleElements = (from el in scaleString.Split(',') select double.Parse(el,
      System.Globalization.CultureInfo.InvariantCulture)).ToArray();
03846
03847                double scaleX = scaleElements[0];
03848                double scaleY = scaleElements.Length > 1 ?  scaleElements[1] :  scaleX;
03849
03850                a = a * scaleX;
03851                b = b * scaleX;
03852                c = c * scaleY;
03853                d = d * scaleY;
03854            }
03855            else
03856            {
03857                throw new ArgumentException("Invalid matrix transform:  " + transformMatrix);
03858            }
03859
03860
03861            double delta = a * d - b * c;
03862
03863            Point translation = new Point(e, f);
03864            double rotation = 0;
03865            Size scale = new Size(0, 0);
03866            Size skew = new Size(0, 0);
03867
03868            if (a != 0 || b != 0)
03869            {
03870                double r = Math.Sqrt(a * a + b * b);
03871                rotation = b > 0 ?  Math.Acos(a / r) :  -Math.Acos(a / r);
03872                scale = new Size(r, delta / r);
03873                new Size(Math.Atan((a * c + b * d) / (r * r)), 0);
03874            }
03875            else if (c != 0 || d != 0)
03876            {
03877                double s = Math.Sqrt(c * c + d * d);
03878                rotation = Math.PI / 2 - (d > 0 ?  Math.Acos(-c / s) :  -Math.Acos(c / s));
03879                scale = new Size(delta / s, s);
03880                skew = new Size(0, Math.Atan((a * c + b * d) / (s * s)));
03881            }
03882            else
03883            {
03884                // a = b = c = d = 0
03885            }
03886
03887            return (translation, rotation, scale, skew);
03888        }
03889
03890        private static Dictionary<string, XmlElement> GetTaggedElements(XmlNode node)
03891        {
03892            Dictionary<string, XmlElement> tbr = new Dictionary<string, XmlElement>();
03893
03894            foreach (XmlNode childNode in node)
03895            {
03896                if (childNode is XmlElement child)
03897                {
03898                    string id = child.GetAttribute("id");
03899                    if (!string.IsNullOrEmpty(id))
03900                    {
03901                        tbr[id] = child;
03902                    }
03903
03904                    foreach (KeyValuePair<string, XmlElement> kvp in GetTaggedElements(child))
03905                    {
03906                        tbr[kvp.Key] = kvp.Value;
03907                    }
03908                }
03909            }
03910
03911            return tbr;
03912        }
03913
```

```
03914         internal static XmlElement ToLinearGradient(this LinearGradientBrush brush, XmlDocument
      document, string gradientId)
03915         {
03916             XmlElement gradient = document.CreateElement("linearGradient", SVGContext.SVGNamespace);
03917 
03918             gradient.SetAttribute("id", gradientId);
03919 
03920             gradient.SetAttribute("gradientUnits", "userSpaceOnUse");
03921 
03922             gradient.SetAttribute("x1",
      brush.StartPoint.X.ToString(System.Globalization.CultureInfo.InvariantCulture));
03923             gradient.SetAttribute("y1",
      brush.StartPoint.Y.ToString(System.Globalization.CultureInfo.InvariantCulture));
03924             gradient.SetAttribute("x2",
      brush.EndPoint.X.ToString(System.Globalization.CultureInfo.InvariantCulture));
03925             gradient.SetAttribute("y2",
      brush.EndPoint.Y.ToString(System.Globalization.CultureInfo.InvariantCulture));
03926 
03927             int index = 0;
03928             foreach (GradientStop stop in brush.GradientStops)
03929             {
03930                 XmlElement gradientStop = document.CreateElement("stop", SVGContext.SVGNamespace);
03931 
03932                 gradientStop.SetAttribute("offset",
      stop.Offset.ToString(System.Globalization.CultureInfo.InvariantCulture));
03933                 gradientStop.SetAttribute("stop-color", stop.Colour.ToCSSString(false));
03934                 gradientStop.SetAttribute("stop-opacity",
      stop.Colour.A.ToString(System.Globalization.CultureInfo.InvariantCulture));
03935                 gradientStop.SetAttribute("id", gradientId + "/stop" + index.ToString());
03936 
03937                 gradient.AppendChild(gradientStop);
03938                 index++;
03939             }
03940 
03941             return gradient;
03942         }
03943 
03944         internal static XmlElement ToRadialGradient(this RadialGradientBrush brush, XmlDocument
      document, string gradientId)
03945         {
03946             XmlElement gradient = document.CreateElement("radialGradient", SVGContext.SVGNamespace);
03947 
03948             gradient.SetAttribute("id", gradientId);
03949 
03950             gradient.SetAttribute("gradientUnits", "userSpaceOnUse");
03951 
03952             gradient.SetAttribute("cx",
      brush.Centre.X.ToString(System.Globalization.CultureInfo.InvariantCulture));
03953             gradient.SetAttribute("cy",
      brush.Centre.Y.ToString(System.Globalization.CultureInfo.InvariantCulture));
03954             gradient.SetAttribute("r",
      brush.Radius.ToString(System.Globalization.CultureInfo.InvariantCulture));
03955             gradient.SetAttribute("fx",
      brush.FocalPoint.X.ToString(System.Globalization.CultureInfo.InvariantCulture));
03956             gradient.SetAttribute("fy",
      brush.FocalPoint.Y.ToString(System.Globalization.CultureInfo.InvariantCulture));
03957 
03958             int index = 0;
03959             foreach (GradientStop stop in brush.GradientStops)
03960             {
03961                 XmlElement gradientStop = document.CreateElement("stop", SVGContext.SVGNamespace);
03962 
03963                 gradientStop.SetAttribute("offset",
      stop.Offset.ToString(System.Globalization.CultureInfo.InvariantCulture));
03964                 gradientStop.SetAttribute("stop-color", stop.Colour.ToCSSString(false));
03965                 gradientStop.SetAttribute("stop-opacity",
      stop.Colour.A.ToString(System.Globalization.CultureInfo.InvariantCulture));
03966                 gradientStop.SetAttribute("id", gradientId + "/stop" + index.ToString());
03967 
03968                 gradient.AppendChild(gradientStop);
03969                 index++;
03970             }
03971 
03972             return gradient;
03973         }
03974 
03975         // Adapted from
      http://www.ericwhite.com/blog/2011/05/09/custom-formatting-of-xml-using-linq-to-xml-2/
03976         private static void WriteStartElement(XmlWriter writer, XmlElement e)
03977         {
03978             writer.WriteStartElement(e.Prefix, e.LocalName, e.NamespaceURI);
03979 
03980             foreach (XmlAttribute a in e.Attributes)
03981             {
03982                 writer.WriteAttributeString(a.Prefix, a.LocalName, a.NamespaceURI, a.Value);
03983             }
03984         }
```

```
03985
03986         private static void WriteElement(XmlWriter writer, XmlElement e)
03987         {
03988             if (e.Name == "text")
03989             {
03990                 XmlWriterSettings settings = new XmlWriterSettings();
03991                 settings.Indent = false;
03992                 settings.OmitXmlDeclaration = true;
03993                 settings.ConformanceLevel = ConformanceLevel.Fragment;
03994                 settings.NamespaceHandling = NamespaceHandling.OmitDuplicates;
03995
03996                 WriteStartElement(writer, e);
03997
03998                 StringBuilder sb = new StringBuilder();
03999
04000                 using (XmlWriter newWriter = XmlWriter.Create(sb, settings))
04001                 {
04002                     foreach (XmlNode n in e.ChildNodes)
04003                     {
04004                         n.WriteTo(newWriter);
04005                     }
04006                 }
04007
04008                 writer.WriteRaw(sb.ToString().Replace(" xmlns=\"http://www.w3.org/2000/svg\">", ">"));
04009
04010                 writer.WriteEndElement();
04011             }
04012             else
04013             {
04014                 WriteStartElement(writer, e);
04015                 foreach (XmlNode n in e.ChildNodes)
04016                 {
04017                     if (n is XmlElement element)
04018                     {
04019                         WriteElement(writer, element);
04020                     }
04021                     else
04022                     {
04023                         n.WriteTo(writer);
04024                     }
04025                 }
04026                 writer.WriteEndElement();
04027             }
04028         }
04029
04030         private static void WriteXMLToStream(XmlElement element, Stream output)
04031         {
04032             XmlWriterSettings settings = new XmlWriterSettings();
04033             settings.Indent = true;
04034
04035             using (XmlWriter writer = XmlWriter.Create(output, settings))
04036             {
04037                 WriteElement(writer, element);
04038             }
04039         }
04040
04041 /// <summary>
04042 /// Given an <see cref="XmlDocument"/> that contains an SVG image, write its contents to the specified
      <paramref name="output"/> <see cref="Stream"/>.
04043 /// </summary>
04044 /// <param name="svgDocument">An <see cref="XmlDocument"/> containing an SVG image</param>
04045 /// <param name="output">The <see cref="Stream"/> on which the document will be written.</param>
04046 /// <remarks>Compared to the built-in methods to save XML documents, this method takes care of some
      quirks of the SVG format.</remarks>
04047         public static void WriteSVGXML(this XmlDocument svgDocument, Stream output)
04048         {
04049             WriteXMLToStream(svgDocument.DocumentElement, output);
04050         }
04051
04052 /// <summary>
04053 /// Given an <see cref="XmlDocument"/> that contains an SVG image, write its contents to the specified
      output file.
04054 /// </summary>
04055 /// <param name="svgDocument">An <see cref="XmlDocument"/> containing an SVG image</param>
04056 /// <param name="outputFile">The file to which the XML document will be saved.</param>
04057 /// <remarks>Compared to the built-in methods to save XML documents, this method takes care of some
      quirks of the SVG format.</remarks>
04058         public static void WriteSVGXML(this XmlDocument svgDocument, string outputFile)
04059         {
04060             using (FileStream fs = new FileStream(outputFile, FileMode.Create))
04061             {
04062                 svgDocument.WriteSVGXML(fs);
04063             }
04064         }
04065
04066 /// <summary>
04067 /// Given an <see cref="XmlDocument"/> that contains an SVG image, write its contents to a <see
```

```
    langword="string"/>.
04068 /// </summary>
04069 /// <param name="svgDocument">An <see cref="XmlDocument"/> containing an SVG image</param>
04070 /// <remarks>Compared to the built-in methods to save XML documents, this method takes care of some
    quirks of the SVG format.</remarks>
04071         public static string WriteSVGXML(this XmlDocument svgDocument)
04072         {
04073             using (MemoryStream ms = new MemoryStream())
04074             {
04075                 svgDocument.WriteSVGXML(ms);
04076                 ms.Seek(0, SeekOrigin.Begin);
04077 
04078                 return Encoding.UTF8.GetString(ms.GetBuffer(), 0, (int)ms.Length);
04079             }
04080         }
04081     }
04082 }
```

8.62 SVGParser.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using ExCSS;
00019 using System;
00020 using System.Collections.Generic;
00021 using System.IO;
00022 using System.Linq;
00023 using System.Text;
00024 using System.Text.RegularExpressions;
00025 using System.Xml;
00026 using VectSharp.Filters;
00027 
00028 namespace VectSharp.SVG
00029 {
00030 /// <summary>
00031 /// Contains methods to read an SVG image file.
00032 /// </summary>
00033     public static class Parser
00034     {
00035         static Parser()
00036         {
00037             ParseImageURI = ParseSVGURI;
00038         }
00039 
00040 /// <summary>
00041 /// A function that takes as input an image URI and a boolean value indicating whether the image
    should be interpolated, and returns a <see cref="Page"/> object containing the image.
00042 /// By default, this is equal to <see cref="ParseSVGURI"/>, i.e.  it is only able to parse SVG images.
    If you wish to enable the parsing of other formats, you should install the "VectSharp.MuPDFUtils"
    NuGet package
00043 /// and enable the parser in your program by doing something like:
00044 /// <code>VectSharp.SVG.Parser.ParseImageURI =
    VectSharp.MuPDFUtils.ImageURIParser.Parser(VectSharp.SVG.Parser.ParseSVGURI);</code>
00045 /// </summary>
00046         public static Func<string, bool, Page> ParseImageURI;
00047 
00048 /// <summary>
00049 /// Parses an SVG image URI.
00050 /// </summary>
00051 /// <param name="uri">The image URI to parse.</param>
00052 /// <param name="ignored">This value is ignored and is only needed for compatibility.</param>
00053 /// <returns>A <see cref="Page"/> containing the parsed SVG image, or null.</returns>
00054         public static Page ParseSVGURI(string uri, bool ignored = false)
00055         {
00056             if (uri.StartsWith("data:"))
00057             {
00058                 string mimeType = uri.Substring(uri.IndexOf(":") + 1, uri.IndexOf(";") -
    uri.IndexOf(":") - 1);
```

```
00059
00060                 string type = uri.Substring(uri.IndexOf(";") + 1, uri.IndexOf(",") - uri.IndexOf(";")
    - 1);
00061
00062                 if (mimeType == "image/svg+xml")
00063                 {
00064                     int offset = uri.IndexOf(",") + 1;
00065
00066                     string data;
00067
00068                     switch (type)
00069                     {
00070                         case "base64":
00071                             data =
    Encoding.UTF8.GetString(Convert.FromBase64String(uri.Substring(offset)));
00072                             break;
00073                         case "":
00074                         case "charset=utf-8":
00075                         case "utf-8":
00076                         case "utf8":
00077                             data = System.Web.HttpUtility.UrlDecode(uri.Substring(offset));
00078                             break;
00079                         case "charset=ascii":
00080                         case "ascii":
00081                             data = System.Web.HttpUtility.UrlDecode(uri.Substring(offset));
00082                             break;
00083                         default:
00084                             throw new InvalidDataException("Unknown data stream type!");
00085                     }
00086
00087                     try
00088                     {
00089                         StringReader sr = new StringReader(data);
00090                         string firstLine = sr.ReadLine();
00091                         sr.Dispose();
00092
00093                         if (firstLine.StartsWith("<?xml") && firstLine.EndsWith("?>"))
00094                         {
00095                             data = data.Substring(firstLine.Length + 1);
00096                         }
00097                     }
00098                     catch { }
00099
00100                     return FromString(data);
00101                 }
00102                 else
00103                 {
00104                     return null;
00105                 }
00106             }
00107
00108             return null;
00109         }
00110
00111 /// <summary>
00112 /// Parses SVG source into a <see cref="Page"/> containing the image represented by the code.
00113 /// </summary>
00114 /// <param name="svgSource">The SVG source code.</param>
00115 /// <returns>A <see cref="Page"/> containing the image represented by the <paramref
    name="svgSource"/>.</returns>
00116         public static Page FromString(string svgSource)
00117         {
00118             XmlDocument svgDoc = new XmlDocument();
00119
00120
00121             svgDoc.LoadXml(svgSource);
00122
00123             Dictionary<string, FontFamily> embeddedFonts = new Dictionary<string, FontFamily>();
00124
00125             StylesheetParser parser = new StylesheetParser();
00126
00127             List<Stylesheet> styleSheets = new List<Stylesheet>();
00128
00129             foreach (XmlNode styleNode in svgDoc.GetElementsByTagName("style"))
00130             {
00131                 foreach (KeyValuePair<string, FontFamily> fnt in
    GetEmbeddedFonts(styleNode.InnerText))
00132                 {
00133                     embeddedFonts.Add(fnt.Key, fnt.Value);
00134                 }
00135
00136                 try
00137                 {
00138                     Stylesheet sheet = parser.Parse(styleNode.InnerText);
00139                     styleSheets.Add(sheet);
00140                 }
00141                 catch { }
```

```
00142             }
00143
00144             Dictionary<string, (bool, Brush)> gradients = new Dictionary<string, (bool, Brush)>();
00145             Dictionary<string, IFilter> filters = new Dictionary<string, IFilter>();
00146             Dictionary<string, XmlNode> masks = new Dictionary<string, XmlNode>();
00147
00148
00149             foreach (KeyValuePair<string, (bool, Brush)> fnt in
      GetGradients(svgDoc.GetElementsByTagName("svg")[0], styleSheets))
00150             {
00151                 gradients.Add(fnt.Key, fnt.Value);
00152             }
00153
00154             foreach (KeyValuePair<string, IFilter> filt in
      GetFilters(svgDoc.GetElementsByTagName("svg")[0], styleSheets))
00155             {
00156                 filters.Add(filt.Key, filt.Value);
00157             }
00158
00159             foreach (KeyValuePair<string, XmlNode> mask in
      GetMasks(svgDoc.GetElementsByTagName("svg")[0], styleSheets))
00160             {
00161                 masks.Add(mask.Key, mask.Value);
00162             }
00163
00164             foreach (XmlNode definitionsNode in svgDoc.GetElementsByTagName("defs"))
00165             {
00166                 foreach (KeyValuePair<string, (bool, Brush)> fnt in GetGradients(definitionsNode,
      styleSheets))
00167                 {
00168                     gradients.Add(fnt.Key, fnt.Value);
00169                 }
00170
00171                 foreach (KeyValuePair<string, IFilter> filt in GetFilters(definitionsNode,
      styleSheets))
00172                 {
00173                     filters.Add(filt.Key, filt.Value);
00174                 }
00175
00176                 foreach (KeyValuePair<string, XmlNode> mask in GetMasks(definitionsNode, styleSheets))
00177                 {
00178                     masks.Add(mask.Key, mask.Value);
00179                 }
00180             }
00181
00182
00183
00184             Graphics gpr = new Graphics();
00185
00186             Size pageSize = InterpretSVGObject(svgDoc.GetElementsByTagName("svg")[0], gpr, new
      PresentationAttributes() { EmbeddedFonts = embeddedFonts }, styleSheets, gradients, filters, masks);
00187
00188             Page pg = new Page(pageSize.Width, pageSize.Height);
00189
00190             pg.Graphics = gpr;
00191
00192             return pg;
00193         }
00194
00195 /// <summary>
00196 /// Parses an SVG image file into a <see cref="Page"/> containing the image.
00197 /// </summary>
00198 /// <param name="fileName">The path to the SVG image file.</param>
00199 /// <returns>A <see cref="Page"/> containing the image represented by the file.</returns>
00200         public static Page FromFile(string fileName)
00201         {
00202             return FromString(File.ReadAllText(fileName));
00203         }
00204
00205 /// <summary>
00206 /// Parses an stream containing SVG source code into a <see cref="Page"/> containing the image
      represented by the code.
00207 /// </summary>
00208 /// <param name="svgSourceStream">The stream containing SVG source code.</param>
00209 /// <returns>A <see cref="Page"/> containing the image represented by the <paramref
      name="svgSourceStream"/>.</returns>
00210         public static Page FromStream(Stream svgSourceStream)
00211         {
00212             using (StreamReader sr = new StreamReader(svgSourceStream))
00213             {
00214                 return FromString(sr.ReadToEnd());
00215             }
00216         }
00217
00218         private static Size InterpretSVGObject(XmlNode svgObject, Graphics gpr, PresentationAttributes
      attributes, IEnumerable<Stylesheet> styleSheets, Dictionary<string, (bool, Brush)> gradients,
      Dictionary<string, IFilter> filters, Dictionary<string, XmlNode> masks)
```

```
00219         {
00220             double[] viewBox = ParseListOfDoubles(svgObject.Attributes?["viewBox"]?.Value);
00221 
00222             double width, height, x, y;
00223 
00224             string widthAttribute = svgObject.Attributes?["width"]?.Value?.Replace("px",
      "")?.Replace("pt", "")?.Replace("mm", "")?.Replace("in", "")?.Replace("cm", "");
00225 
00226             if (!double.TryParse(widthAttribute, System.Globalization.NumberStyles.Any,
      System.Globalization.CultureInfo.InvariantCulture, out width)) { width = double.NaN; }
00227 
00228             string heightAttribute = svgObject.Attributes?["height"]?.Value?.Replace("px",
      "")?.Replace("pt", "")?.Replace("mm", "")?.Replace("in", "")?.Replace("cm", "");
00229             if (!double.TryParse(heightAttribute, System.Globalization.NumberStyles.Any,
      System.Globalization.CultureInfo.InvariantCulture, out height)) { height = double.NaN; }
00230 
00231             string xAttribute = svgObject.Attributes?["x"]?.Value;
00232             double.TryParse(xAttribute, System.Globalization.NumberStyles.Any,
      System.Globalization.CultureInfo.InvariantCulture, out x);
00233 
00234             string yAttribute = svgObject.Attributes?["y"]?.Value;
00235             double.TryParse(yAttribute, System.Globalization.NumberStyles.Any,
      System.Globalization.CultureInfo.InvariantCulture, out y);
00236 
00237             double scaleX = 1;
00238             double scaleY = 1;
00239 
00240             double postTranslateX = 0;
00241             double postTranslateY = 0;
00242 
00243             if (viewBox != null)
00244             {
00245                 if (!double.IsNaN(width) && !double.IsNaN(height))
00246                 {
00247                     scaleX = width / viewBox[2];
00248                     scaleY = height / viewBox[3];
00249                 }
00250                 else if (!double.IsNaN(width) && double.IsNaN(height))
00251                 {
00252                     scaleX = width / viewBox[2];
00253                     scaleY = scaleX;
00254                     height = scaleY * viewBox[3];
00255                 }
00256                 else if (double.IsNaN(width) && !double.IsNaN(height))
00257                 {
00258                     scaleY = height / viewBox[3];
00259                     scaleX = scaleY;
00260                     width = scaleX * viewBox[2];
00261                 }
00262                 else if (double.IsNaN(width) && double.IsNaN(height))
00263                 {
00264                     width = viewBox[2];
00265                     height = viewBox[3];
00266                 }
00267 
00268                 postTranslateX = -viewBox[0];
00269                 postTranslateY = -viewBox[1];
00270             }
00271             else
00272             {
00273                 viewBox = new double[4];
00274 
00275                 if (!double.IsNaN(width))
00276                 {
00277                     viewBox[2] = width;
00278                 }
00279 
00280                 if (!double.IsNaN(height))
00281                 {
00282                     viewBox[3] = height;
00283                 }
00284             }
00285 
00286             double diagonal = Math.Sqrt(viewBox[2] * viewBox[2] + viewBox[3] * viewBox[3]) /
      Math.Sqrt(2);
00287 
00288             Size tbrSize = new Size(width, height);
00289 
00290             gpr.Save();
00291             gpr.Translate(x, y);
00292             gpr.Scale(scaleX, scaleY);
00293             gpr.Translate(postTranslateX, postTranslateY);
00294 
00295             attributes = InterpretPresentationAttributes(svgObject, attributes, viewBox[2],
      viewBox[3], diagonal, gpr, styleSheets, gradients);
00296 
00297             foreach (KeyValuePair<string, XmlNode> mask in masks)
```

```
00298             {
00299                 Graphics maskGpr = new Graphics();
00300                 InterpretGObject(mask.Value, maskGpr, viewBox[2], viewBox[3], diagonal, attributes,
      styleSheets, gradients, filters);
00301
00302                 filters.Add(mask.Key, new MaskFilter(maskGpr));
00303             }
00304
00305             InterpretSVGChildren(svgObject, gpr, attributes, viewBox[2], viewBox[3], diagonal,
      styleSheets, gradients, filters);
00306
00307             gpr.Restore();
00308
00309             return tbrSize;
00310         }
00311
00312         private static void InterpretSVGChildren(XmlNode svgObject, Graphics gpr,
      PresentationAttributes attributes, double width, double height, double diagonal,
      IEnumerable<Stylesheet> styleSheets, Dictionary<string, (bool, Brush)> gradients, Dictionary<string,
      IFilter> filters)
00313         {
00314             foreach (XmlNode child in svgObject.ChildNodes)
00315             {
00316                 InterpretSVGElement(child, gpr, attributes, width, height, diagonal, styleSheets,
      gradients, filters);
00317             }
00318         }
00319
00320         private static void InterpretSVGElement(XmlNode currObject, Graphics gpr,
      PresentationAttributes attributes, double width, double height, double diagonal,
      IEnumerable<Stylesheet> styleSheets, Dictionary<string, (bool, Brush)> gradients, Dictionary<string,
      IFilter> filters)
00321         {
00322             if (currObject.NodeType == XmlNodeType.EntityReference)
00323             {
00324                 InterpretSVGChildren(currObject, gpr, attributes, width, height, diagonal,
      styleSheets, gradients, filters);
00325             }
00326             else if (currObject.Name.Equals("svg", StringComparison.OrdinalIgnoreCase))
00327             {
00328                 InterpretSVGObject(currObject, gpr, attributes, styleSheets, gradients, filters, new
      Dictionary<string, XmlNode>());
00329             }
00330             else if (currObject.Name.Equals("line", StringComparison.OrdinalIgnoreCase))
00331             {
00332                 InterpretLineObject(currObject, gpr, width, height, diagonal, attributes, styleSheets,
      gradients);
00333             }
00334             else if (currObject.Name.Equals("circle", StringComparison.OrdinalIgnoreCase))
00335             {
00336                 InterpretCircleObject(currObject, gpr, width, height, diagonal, attributes,
      styleSheets, gradients);
00337             }
00338             else if (currObject.Name.Equals("ellipse", StringComparison.OrdinalIgnoreCase))
00339             {
00340                 InterpretEllipseObject(currObject, gpr, width, height, diagonal, attributes,
      styleSheets, gradients);
00341             }
00342             else if (currObject.Name.Equals("path", StringComparison.OrdinalIgnoreCase))
00343             {
00344                 InterpretPathObject(currObject, gpr, width, height, diagonal, attributes, styleSheets,
      gradients);
00345             }
00346             else if (currObject.Name.Equals("polyline", StringComparison.OrdinalIgnoreCase))
00347             {
00348                 InterpretPolyLineObject(currObject, false, gpr, width, height, diagonal, attributes,
      styleSheets, gradients);
00349             }
00350             else if (currObject.Name.Equals("polygon", StringComparison.OrdinalIgnoreCase))
00351             {
00352                 InterpretPolyLineObject(currObject, true, gpr, width, height, diagonal, attributes,
      styleSheets, gradients);
00353             }
00354             else if (currObject.Name.Equals("rect", StringComparison.OrdinalIgnoreCase))
00355             {
00356                 InterpretRectObject(currObject, gpr, width, height, diagonal, attributes, styleSheets,
      gradients);
00357             }
00358             else if (currObject.Name.Equals("use", StringComparison.OrdinalIgnoreCase))
00359             {
00360                 InterpretUseObject(currObject, gpr, width, height, diagonal, attributes, styleSheets,
      gradients, filters);
00361             }
00362             else if (currObject.Name.Equals("g", StringComparison.OrdinalIgnoreCase) ||
      currObject.Name.Equals("symbol", StringComparison.OrdinalIgnoreCase))
00363             {
00364                 InterpretGObject(currObject, gpr, width, height, diagonal, attributes, styleSheets,
```

```
gradients, filters);
00365             }
00366             else if (currObject.Name.Equals("text", StringComparison.OrdinalIgnoreCase))
00367             {
00368                 double x = 0;
00369                 double y = 0;
00370 
00371                 InterpretTextObject(currObject, gpr, width, height, diagonal, attributes, styleSheets,
gradients, ref x, ref y);
00372             }
00373             else if (currObject.Name.Equals("image", StringComparison.OrdinalIgnoreCase))
00374             {
00375                 InterpretImageObject(currObject, gpr, width, height, diagonal, attributes,
styleSheets, gradients);
00376             }
00377         }
00378 
00379         private static void InterpretImageObject(XmlNode currObject, Graphics gpr, double width,
double height, double diagonal, PresentationAttributes attributes, IEnumerable<Stylesheet>
styleSheets, Dictionary<string, (bool, Brush)> gradients)
00380         {
00381             PresentationAttributes currAttributes = InterpretPresentationAttributes(currObject,
attributes, width, height, diagonal, gpr, styleSheets, gradients);
00382 
00383             double x = ParseLengthOrPercentage(currObject.Attributes?["x"]?.Value, width,
currAttributes.X);
00384             double y = ParseLengthOrPercentage(currObject.Attributes?["y"]?.Value, height,
currAttributes.Y);
00385 
00386             double w = ParseLengthOrPercentage(currObject.Attributes?["width"]?.Value, width,
currAttributes.Width);
00387             double h = ParseLengthOrPercentage(currObject.Attributes?["height"]?.Value, height,
currAttributes.Height);
00388 
00389             bool interpolate = !(currObject.Attributes?["image-rendering"]?.Value == "pixelated" ||
currObject.Attributes?["image-rendering"]?.Value == "optimizeSpeed");
00390 
00391             string href = currObject.Attributes?["href"]?.Value;
00392 
00393             if (string.IsNullOrEmpty(href))
00394             {
00395                 href = currObject.Attributes?["xlink:href"]?.Value;
00396             }
00397 
00398             bool hadClippingPath = ApplyClipPath(currObject, gpr, width, height, diagonal, attributes,
styleSheets, gradients);
00399 
00400             string tag = currObject.Attributes?["id"]?.Value;
00401 
00402             if (!string.IsNullOrEmpty(href) && w > 0 && h > 0)
00403             {
00404                 Page image = ParseImageURI(href, interpolate);
00405 
00406                 if (image != null)
00407                 {
00408                     gpr.Save();
00409 
00410                     double scaleX = w / image.Width;
00411                     double scaleY = h / image.Height;
00412 
00413                     gpr.Scale(scaleX, scaleY);
00414 
00415                     gpr.DrawGraphics(x / scaleX, y / scaleY, image.Graphics, tag:  tag);
00416 
00417                     gpr.Restore();
00418                 }
00419                 else
00420                 {
00421                     gpr.StrokeRectangle(x, y, w, h, Colours.Red, 0.1, tag:  tag);
00422                     gpr.StrokePath(new GraphicsPath().MoveTo(x, y).LineTo(x + w, y + h).MoveTo(x + w,
y).LineTo(x, y + h), Colours.Red, 0.1, tag:  tag);
00423                 }
00424             }
00425 
00426             if (hadClippingPath)
00427             {
00428                 gpr.Restore();
00429             }
00430 
00431             if (currAttributes.NeedsRestore)
00432             {
00433                 gpr.Restore();
00434             }
00435         }
00436 
00437         private static string GetFirstAttributeValueIncludingAncestors(XmlNode currObject, string
attribute)
```

```
00438         {
00439             string tbr = currObject.Attributes?[attribute]?.Value;
00440 
00441             if (tbr != null)
00442             {
00443                 return tbr;
00444             }
00445             else
00446             {
00447                 if (currObject.ParentNode != null && !currObject.ParentNode.Name.Equals("svg",
      StringComparison.OrdinalIgnoreCase))
00448                 {
00449                     return GetFirstAttributeValueIncludingAncestors(currObject.ParentNode, attribute);
00450                 }
00451                 else
00452                 {
00453                     return null;
00454                 }
00455             }
00456         }
00457 
00458         private static void InterpretTextObject(XmlNode currObject, Graphics gpr, double width, double
      height, double diagonal, PresentationAttributes attributes, IEnumerable<Stylesheet> styleSheets,
      Dictionary<string, (bool, Brush)> gradients, ref double x, ref double y, double fontSize = double.NaN,
      string fontFamily = null, string textAlign = null)
00459         {
00460             PresentationAttributes currAttributes = InterpretPresentationAttributes(currObject,
      attributes, width, height, diagonal, gpr, styleSheets, gradients);
00461 
00462             x = ParseLengthOrPercentage(currObject.Attributes?["x"]?.Value, width, x);
00463             y = ParseLengthOrPercentage(currObject.Attributes?["y"]?.Value, height, y);
00464 
00465             double dx = ParseLengthOrPercentage(currObject.Attributes?["dx"]?.Value, width, 0);
00466             double dy = ParseLengthOrPercentage(currObject.Attributes?["dy"]?.Value, height, 0);
00467 
00468             x += dx;
00469             y += dy;
00470 
00471             fontFamily = GetFirstAttributeValueIncludingAncestors(currObject, "font-family") ??
      fontFamily;
00472             fontSize = ParseLengthOrPercentage(GetFirstAttributeValueIncludingAncestors(currObject,
      "font-size"), width, fontSize);
00473             textAlign = GetFirstAttributeValueIncludingAncestors(currObject, "text-align") ??
      GetFirstAttributeValueIncludingAncestors(currObject, "text-anchor") ??  textAlign;
00474             if (textAlign == "start")
00475             {
00476                 textAlign = "left";
00477             }
00478             else if (textAlign == "middle")
00479             {
00480                 textAlign = "center";
00481             }
00482             else if (textAlign == "end")
00483             {
00484                 textAlign = "right";
00485             }
00486 
00487             bool hadClippingPath = ApplyClipPath(currObject, gpr, width, height, diagonal, attributes,
      styleSheets, gradients);
00488 
00489             string tag = currObject.Attributes?["id"]?.Value;
00490 
00491             if (currObject.ChildNodes.OfType<XmlNode>().Any(a => a.NodeType != XmlNodeType.Text))
00492             {
00493                 foreach (XmlNode child in currObject.ChildNodes)
00494                 {
00495                     InterpretTextObject(child, gpr, width, height, diagonal, currAttributes,
      styleSheets, gradients, ref x, ref y, fontSize, fontFamily, textAlign);
00496                 }
00497 
00498                 if (hadClippingPath)
00499                 {
00500                     gpr.Restore();
00501                 }
00502 
00503                 if (currAttributes.NeedsRestore)
00504                 {
00505                     gpr.Restore();
00506                 }
00507             }
00508             else
00509             {
00510                 string text = currObject.InnerText;
00511 
00512                 if (!double.IsNaN(fontSize) && !string.IsNullOrEmpty(text))
00513                 {
00514                     text = text.Replace("\u00A0", " ");
```

```
00515
00516                   FontFamily parsedFontFamily = ParseFontFamily(fontFamily,
    currAttributes.EmbeddedFonts);
00517                   string fontWeight = GetFirstAttributeValueIncludingAncestors(currObject,
    "font-weight");
00518                   string fontStyle = GetFirstAttributeValueIncludingAncestors(currObject,
    "font-style");
00519
00520                   bool isBold = false;
00521                   bool isItalic = false;
00522
00523                   if (fontWeight != null && (fontWeight.Equals("bold",
    StringComparison.OrdinalIgnoreCase) || fontWeight.Equals("bolder", StringComparison.OrdinalIgnoreCase)
    || (int.TryParse(fontWeight, out int weight) && weight >= 500)))
00524                   {
00525                       isBold = true;
00526                   }
00527
00528                   if (fontStyle != null && (fontStyle.Equals("italic",
    StringComparison.OrdinalIgnoreCase) || fontStyle.Equals("oblique",
    StringComparison.OrdinalIgnoreCase)))
00529                   {
00530                       isItalic = true;
00531                   }
00532
00533                   if (isBold && !isItalic)
00534                   {
00535                       parsedFontFamily = GetBoldFontFamily(parsedFontFamily);
00536                   }
00537                   else if (isItalic && !isBold)
00538                   {
00539                       parsedFontFamily = GetItalicFontFamily(parsedFontFamily);
00540                   }
00541                   else if (isItalic && isBold)
00542                   {
00543                       parsedFontFamily = GetBoldItalicFontFamily(parsedFontFamily);
00544                   }
00545
00546
00547                   Font fnt = new Font(parsedFontFamily, fontSize);
00548
00549                   double endX = x;
00550
00551                   if (fnt.FontFamily.TrueTypeFile != null)
00552                   {
00553                       Font.DetailedFontMetrics metrics = fnt.MeasureTextAdvanced(text);
00554                       x += metrics.LeftSideBearing;
00555
00556                       if (!string.IsNullOrEmpty(textAlign) && (textAlign.Equals("right",
    StringComparison.OrdinalIgnoreCase) || textAlign.Equals("end", StringComparison.OrdinalIgnoreCase)))
00557                       {
00558                           x -= metrics.AdvanceWidth;
00559                       }
00560                       else if (!string.IsNullOrEmpty(textAlign) && textAlign.Equals("center",
    StringComparison.OrdinalIgnoreCase))
00561                       {
00562                           x -= metrics.AdvanceWidth * 0.5;
00563                       }
00564
00565                       endX += metrics.AdvanceWidth;
00566                   }
00567
00568                   TextBaselines baseline = TextBaselines.Baseline;
00569
00570                   string textBaseline = GetFirstAttributeValueIncludingAncestors(currObject,
    "alignment-baseline");
00571
00572                   if (textBaseline != null)
00573                   {
00574                       if (textBaseline.Equals("text-bottom", StringComparison.OrdinalIgnoreCase) ||
    textBaseline.Equals("bottom", StringComparison.OrdinalIgnoreCase))
00575                       {
00576                           baseline = TextBaselines.Bottom;
00577                       }
00578                       if (textBaseline.Equals("middle", StringComparison.OrdinalIgnoreCase) ||
    textBaseline.Equals("central", StringComparison.OrdinalIgnoreCase) || textBaseline.Equals("center",
    StringComparison.OrdinalIgnoreCase))
00579                       {
00580                           baseline = TextBaselines.Middle;
00581                       }
00582                       if (textBaseline.Equals("text-top", StringComparison.OrdinalIgnoreCase) ||
    textBaseline.Equals("top", StringComparison.OrdinalIgnoreCase) || textBaseline.Equals("hanging",
    StringComparison.OrdinalIgnoreCase))
00583                       {
00584                           baseline = TextBaselines.Top;
00585                       }
00586                   }
```

```
00587
00588                   if (currAttributes.StrokeFirst)
00589                   {
00590                       if (currAttributes.Stroke != null)
00591                       {
00592                           Brush strokeColour =
    currAttributes.Stroke.MultiplyOpacity(currAttributes.Opacity * currAttributes.StrokeOpacity);
00593
00594                           double[,] transform = null;
00595
00596                           if (currAttributes.StrokeGradientNeedsTransform)
00597                           {
00598                               (strokeColour, transform) = TransformBrush(strokeColour, new
    GraphicsPath().AddText(x, y, text, fnt, baseline).GetBounds());
00599                           }
00600
00601                           if (transform != null)
00602                           {
00603                               GraphicsPath pth = new GraphicsPath().AddText(x, y, text, fnt,
    baseline);
00604                               pth = pth.Transform(MatrixUtils.GetInverseTransformation(transform));
00605
00606                               gpr.Save();
00607                               gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
    transform[1, 1], transform[0, 2], transform[1, 2]);
00608                               gpr.StrokePath(pth, strokeColour, currAttributes.StrokeThickness,
    currAttributes.LineCap, currAttributes.LineJoin, currAttributes.LineDash, tag:  tag);
00609                               gpr.Restore();
00610                           }
00611                           else
00612                           {
00613                               gpr.StrokeText(x, y, text, fnt, strokeColour, baseline,
    currAttributes.StrokeThickness, currAttributes.LineCap, currAttributes.LineJoin,
    currAttributes.LineDash, tag:  tag);
00614                           }
00615                       }
00616
00617                       if (currAttributes.Fill != null)
00618                       {
00619                           Brush fillColour =
    currAttributes.Fill.MultiplyOpacity(currAttributes.Opacity * currAttributes.FillOpacity);
00620
00621                           double[,] transform = null;
00622
00623                           if (currAttributes.FillGradientNeedsTransform)
00624                           {
00625                               (fillColour, transform) = TransformBrush(fillColour, new
    GraphicsPath().AddText(x, y, text, fnt, baseline).GetBounds());
00626                           }
00627
00628                           if (transform != null)
00629                           {
00630                               GraphicsPath pth = new GraphicsPath().AddText(x, y, text, fnt,
    baseline);
00631                               pth = pth.Transform(MatrixUtils.GetInverseTransformation(transform));
00632
00633                               gpr.Save();
00634                               gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
    transform[1, 1], transform[0, 2], transform[1, 2]);
00635                               gpr.FillPath(pth, fillColour, tag:  tag);
00636                               gpr.Restore();
00637                           }
00638                           else
00639                           {
00640                               gpr.FillText(x, y, text, fnt, fillColour, baseline, tag:  tag);
00641                           }
00642                       }
00643                   }
00644                   else
00645                   {
00646                       if (currAttributes.Fill != null)
00647                       {
00648                           Brush fillColour =
    currAttributes.Fill.MultiplyOpacity(currAttributes.Opacity * currAttributes.FillOpacity);
00649
00650                           double[,] transform = null;
00651
00652                           if (currAttributes.FillGradientNeedsTransform)
00653                           {
00654                               (fillColour, transform) = TransformBrush(fillColour, new
    GraphicsPath().AddText(x, y, text, fnt, baseline).GetBounds());
00655                           }
00656
00657                           if (transform != null)
00658                           {
00659                               GraphicsPath pth = new GraphicsPath().AddText(x, y, text, fnt,
    baseline);
```

```
00660                                   pth = pth.Transform(MatrixUtils.GetInverseTransformation(transform));
00661
00662                                   gpr.Save();
00663                                   gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
00664                                   gpr.FillPath(pth, fillColour, tag:  tag);
00665                                   gpr.Restore();
00666                               }
00667                               else
00668                               {
00669                                   gpr.FillText(x, y, text, fnt, fillColour, baseline, tag:  tag);
00670                               }
00671                           }
00672
00673                           if (currAttributes.Stroke != null)
00674                           {
00675                               Brush strokeColour =
      currAttributes.Stroke.MultiplyOpacity(currAttributes.Opacity * currAttributes.StrokeOpacity);
00676                               double[,] transform = null;
00677
00678                               if (currAttributes.StrokeGradientNeedsTransform)
00679                               {
00680                                   (strokeColour, transform) = TransformBrush(strokeColour, new
      GraphicsPath().AddText(x, y, text, fnt, baseline).GetBounds());
00681                               }
00682
00683                               if (transform != null)
00684                               {
00685                                   GraphicsPath pth = new GraphicsPath().AddText(x, y, text, fnt,
      baseline);
00686                                   pth = pth.Transform(MatrixUtils.GetInverseTransformation(transform));
00687
00688                                   gpr.Save();
00689                                   gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
00690                                   gpr.StrokePath(pth, strokeColour, currAttributes.StrokeThickness,
      currAttributes.LineCap, currAttributes.LineJoin, currAttributes.LineDash, tag:  tag);
00691                                   gpr.Restore();
00692                               }
00693                               else
00694                               {
00695                                   gpr.StrokeText(x, y, text, fnt, strokeColour, baseline,
      currAttributes.StrokeThickness, currAttributes.LineCap, currAttributes.LineJoin,
      currAttributes.LineDash, tag:  tag);
00696                               }
00697                           }
00698                       }
00699
00700                       x = endX;
00701                   }
00702
00703                   if (hadClippingPath)
00704                   {
00705                       gpr.Restore();
00706                   }
00707
00708                   if (currAttributes.NeedsRestore)
00709                   {
00710                       gpr.Restore();
00711                   }
00712               }
00713           }
00714
00715           private static (Brush, double[,]) TransformBrush(Brush brush, Rectangle boundingBox)
00716           {
00717               if (brush is SolidColourBrush)
00718               {
00719                   return (brush, null);
00720               }
00721               else if (brush is LinearGradientBrush linear)
00722               {
00723                   double[,] transformMatrix = MatrixUtils.Identity;
00724                   transformMatrix = MatrixUtils.Translate(transformMatrix, boundingBox.Location.X,
      boundingBox.Location.Y);
00725                   transformMatrix = MatrixUtils.Scale(transformMatrix, boundingBox.Size.Width,
      boundingBox.Size.Height);
00726
00727                   //LinearGradientBrush tbr = new
      LinearGradientBrush(MatrixUtils.Multiply(transformMatrix, linear.StartPoint),
      MatrixUtils.Multiply(transformMatrix, linear.EndPoint), linear.GradientStops);
00728                   return (brush, transformMatrix);
00729               }
00730               else if (brush is RadialGradientBrush radial)
00731               {
00732                   double[,] transformMatrix = MatrixUtils.Identity;
00733                   transformMatrix = MatrixUtils.Translate(transformMatrix, boundingBox.Location.X,
      boundingBox.Location.Y);
```

```
00734                 transformMatrix = MatrixUtils.Scale(transformMatrix, boundingBox.Size.Width,
      boundingBox.Size.Height);
00735
00736                 double determinant = transformMatrix[0, 0] * (transformMatrix[1, 1] *
      transformMatrix[2, 2] - transformMatrix[1, 2] * transformMatrix[2, 1]) -
00737                             transformMatrix[0, 1] * (transformMatrix[1, 0] * transformMatrix[2, 2] -
      transformMatrix[1, 2] * transformMatrix[2, 0]) +
00738                             transformMatrix[0, 2] * (transformMatrix[1, 0] * transformMatrix[2, 1] -
      transformMatrix[1, 1] * transformMatrix[2, 0]);
00739
00740                 //RadialGradientBrush tbr = new
      RadialGradientBrush(MatrixUtils.Multiply(transformMatrix, radial.FocalPoint),
      MatrixUtils.Multiply(transformMatrix, radial.Centre), radial.Radius * Math.Sqrt(determinant),
      radial.GradientStops);
00741
00742                 return (brush, transformMatrix);
00743             }
00744             else
00745             {
00746                 return (brush, null);
00747             }
00748
00749         }
00750
00751         private static string[] BoldPredicates = new string[] { "-Bold", "-bold", " Bold", " bold" };
00752         private static string[] ItalicPredicates = new string[] { "-Italic", "-italic", " Italic", "
      italic", "-Oblique", "-oblique", " Oblique", " oblique" };
00753         private static string[] BoldItalicPredicates = new string[] { "-BoldItalic", "-bolditalic", "
      BoldItalic", " bolditalic", " Bold Italic", " bold italic", "-BoldOblique", "-boldoblique", "
      BoldOblique", " boldoblique", " Bold Oblique", " bold oblique" };
00754
00755         private static FontFamily GetBoldFontFamily(FontFamily fontFamily)
00756         {
00757             switch (fontFamily.FileName)
00758             {
00759                 case "Times-Roman":
00760                 case "Times-Bold":
00761                     return FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesBold);
00762                 case "Times-Italic":
00763                 case "Times-BoldItalic":
00764                     return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesBoldItalic);
00765                 case "Helvetica":
00766                 case "Helvetica-Bold":
00767                     return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold);
00768                 case "Helvetica-Oblique":
00769                 case "Helvetica-BoldOblique":
00770                     return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBoldOblique);
00771                 case "Courier":
00772                 case "Courier-Bold":
00773                     return FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierBold);
00774                 case "Courier-Oblique":
00775                 case "Courier-BoldOblique":
00776                     return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierBoldOblique);
00777                 default:
00778                     foreach (string sr in BoldPredicates)
00779                     {
00780                         FontFamily attempt = FontFamily.ResolveFontFamily(fontFamily.FamilyName + sr);
00781                         if (attempt != null && attempt.TrueTypeFile != null)
00782                         {
00783                             return attempt;
00784                         }
00785                         attempt = FontFamily.ResolveFontFamily(fontFamily.FileName + sr);
00786                         if (attempt != null && attempt.TrueTypeFile != null)
00787                         {
00788                             return attempt;
00789                         }
00790                     }
00791                     return fontFamily;
00792             }
00793         }
00794
00795         private static FontFamily GetItalicFontFamily(FontFamily fontFamily)
00796         {
00797             switch (fontFamily.FileName)
00798             {
00799                 case "Times-Roman":
00800                 case "Times-Italic":
00801                     return FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesItalic);
00802                 case "Times-Bold":
00803                 case "Times-BoldItalic":
00804                     return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesBoldItalic);
00805                 case "Helvetica":
```

```
00806                 case "Helvetica-Oblique":
00807                     return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaOblique);
00808                 case "Helvetica-Bold":
00809                 case "Helvetica-BoldOblique":
00810                     return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBoldOblique);
00811                 case "Courier":
00812                 case "Courier-Oblique":
00813                     return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierOblique);
00814                 case "Courier-Bold":
00815                 case "Courier-BoldOblique":
00816                     return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierBoldOblique);
00817                 default:
00818                     foreach (string sr in ItalicPredicates)
00819                     {
00820                         FontFamily attempt = FontFamily.ResolveFontFamily(fontFamily.FamilyName + sr);
00821                         if (attempt != null && attempt.TrueTypeFile != null)
00822                         {
00823                             return attempt;
00824                         }
00825                         attempt = FontFamily.ResolveFontFamily(fontFamily.FileName + sr);
00826                         if (attempt != null && attempt.TrueTypeFile != null)
00827                         {
00828                             return attempt;
00829                         }
00830                     }
00831                     return fontFamily;
00832             }
00833         }
00834
00835         private static FontFamily GetBoldItalicFontFamily(FontFamily fontFamily)
00836         {
00837             switch (fontFamily.FileName)
00838             {
00839                 case "Times-Roman":
00840                 case "Times-Italic":
00841                     return FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesItalic);
00842                 case "Times-Bold":
00843                 case "Times-BoldItalic":
00844                     return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesBoldItalic);
00845                 case "Helvetica":
00846                 case "Helvetica-Oblique":
00847                     return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaOblique);
00848                 case "Helvetica-Bold":
00849                 case "Helvetica-BoldOblique":
00850                     return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBoldOblique);
00851                 case "Courier":
00852                 case "Courier-Oblique":
00853                     return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierOblique);
00854                 case "Courier-Bold":
00855                 case "Courier-BoldOblique":
00856                     return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierBoldOblique);
00857                 default:
00858                     foreach (string sr in BoldItalicPredicates)
00859                     {
00860                         FontFamily attempt = FontFamily.ResolveFontFamily(fontFamily.FamilyName + sr);
00861                         if (attempt != null && attempt.TrueTypeFile != null)
00862                         {
00863                             return attempt;
00864                         }
00865                         attempt = FontFamily.ResolveFontFamily(fontFamily.FileName + sr);
00866                         if (attempt != null && attempt.TrueTypeFile != null)
00867                         {
00868                             return attempt;
00869                         }
00870                     }
00871                     return fontFamily;
00872             }
00873         }
00874
00875         private static FontFamily ParseFontFamily(string fontFamily, Dictionary<string, FontFamily>
      embeddedFonts)
00876         {
00877             string[] fontFamilies = Regexes.FontFamilySeparator.Split(fontFamily);
00878
00879             foreach (string fam in fontFamilies)
00880             {
00881                 string family = fam.Trim().Trim(',', '"').Trim();
00882
```

```
00883                 if (embeddedFonts.TryGetValue(family, out FontFamily tbr))
00884                 {
00885                     return tbr;
00886                 }
00887 
00888                 List<(string, int)> matchedFamilies = new List<(string, int)>();
00889 
00890                 for (int i = 0; i < FontFamily.StandardFamilies.Length; i++)
00891                 {
00892                     if (family.StartsWith(FontFamily.StandardFamilies[i]))
00893                     {
00894                         matchedFamilies.Add((FontFamily.StandardFamilies[i],
      FontFamily.StandardFamilies[i].Length));
00895                     }
00896                 }
00897 
00898                 if (matchedFamilies.Count > 0)
00899                 {
00900                     return FontFamily.ResolveFontFamily((from el in matchedFamilies orderby el.Item2
      descending select el.Item1).First());
00901                 }
00902                 else
00903                 {
00904                     if (family.Equals("serif", StringComparison.OrdinalIgnoreCase))
00905                     {
00906                         return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesRoman);
00907                     }
00908                     else if (family.Equals("sans-serif", StringComparison.OrdinalIgnoreCase))
00909                     {
00910                         return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica);
00911                     }
00912                     else if (family.Equals("monospace", StringComparison.OrdinalIgnoreCase))
00913                     {
00914                         return FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Courier);
00915                     }
00916                     else if (family.Equals("cursive", StringComparison.OrdinalIgnoreCase))
00917                     {
00918                         return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesItalic);
00919                     }
00920                     else if (family.Equals("system-ui", StringComparison.OrdinalIgnoreCase))
00921                     {
00922                         return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica);
00923                     }
00924                     else if (family.Equals("ui-serif", StringComparison.OrdinalIgnoreCase))
00925                     {
00926                         return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesRoman);
00927                     }
00928                     else if (family.Equals("ui-sans-serif", StringComparison.OrdinalIgnoreCase))
00929                     {
00930                         return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica);
00931                     }
00932                     else if (family.Equals("ui-monospace", StringComparison.OrdinalIgnoreCase))
00933                     {
00934                         return FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Courier);
00935                     }
00936                     else if (family.Equals("StandardSymbolsPS", StringComparison.OrdinalIgnoreCase))
00937                     {
00938                         return FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Symbol);
00939                     }
00940                     else if (family.Equals("D050000L", StringComparison.OrdinalIgnoreCase))
00941                     {
00942                         return
      FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.ZapfDingbats);
00943                     }
00944                 }
00945 
00946                 FontFamily parsedFamily = FontFamily.ResolveFontFamily(family);
00947 
00948                 if (parsedFamily != null && parsedFamily.TrueTypeFile != null)
00949                 {
00950                     return parsedFamily;
00951                 }
00952             }
00953 
00954             return FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica);
00955         }
00956 
00957         private static void InterpretGObject(XmlNode currObject, Graphics gpr, double width, double
      height, double diagonal, PresentationAttributes attributes, IEnumerable<Stylesheet> styleSheets,
      Dictionary<string, (bool, Brush)> gradients, Dictionary<string, IFilter> filters)
00958         {
```

```
00959             PresentationAttributes currAttributes = InterpretPresentationAttributes(currObject,
      attributes, width, height, diagonal, gpr, styleSheets, gradients);
00960
00961             bool hadClippingPath = ApplyClipPath(currObject, gpr, width, height, diagonal, attributes,
      styleSheets, gradients);
00962
00963             string filter = currObject.Attributes?["filter"]?.Value ??
      currObject.Attributes?["mask"]?.Value;
00964
00965             if (!string.IsNullOrEmpty(filter) && filter.StartsWith("url(#"))
00966             {
00967                 filter = filter.Substring(5, filter.Length - 6);
00968             }
00969
00970             if (!string.IsNullOrEmpty(filter) && filters.ContainsKey(filter))
00971             {
00972                 Graphics filteredGraphics = new Graphics();
00973
00974                 InterpretSVGChildren(currObject, filteredGraphics, currAttributes, width, height,
      diagonal, styleSheets, gradients, filters);
00975                 gpr.DrawGraphics(0, 0, filteredGraphics, filters[filter]);
00976             }
00977             else
00978             {
00979                 InterpretSVGChildren(currObject, gpr, currAttributes, width, height, diagonal,
      styleSheets, gradients, filters);
00980             }
00981
00982             if (hadClippingPath)
00983             {
00984                 gpr.Restore();
00985             }
00986
00987             if (currAttributes.NeedsRestore)
00988             {
00989                 gpr.Restore();
00990             }
00991         }
00992
00993         private static void InterpretUseObject(XmlNode currObject, Graphics gpr, double width, double
      height, double diagonal, PresentationAttributes attributes, IEnumerable<Stylesheet> styleSheets,
      Dictionary<string, (bool, Brush)> gradients, Dictionary<string, IFilter> filters)
00994         {
00995             double x, y, w, h;
00996
00997             x = ParseLengthOrPercentage(currObject.Attributes?["x"]?.Value, width);
00998             y = ParseLengthOrPercentage(currObject.Attributes?["y"]?.Value, height);
00999             w = ParseLengthOrPercentage(currObject.Attributes?["width"]?.Value, width, double.NaN);
01000             h = ParseLengthOrPercentage(currObject.Attributes?["height"]?.Value, height, double.NaN);
01001
01002             string id = currObject.Attributes?["href"]?.Value ??
      currObject.Attributes?["xlink:href"]?.Value;
01003
01004             if (id != null && id.StartsWith("#"))
01005             {
01006                 id = id.Substring(1);
01007
01008                 XmlNode element =
      currObject.OwnerDocument.SelectSingleNode(string.Format("//*[@id='{0}']", id));
01009
01010                 if (element != null)
01011                 {
01012                     XmlNode clone = element.Clone();
01013
01014                     currObject.AppendChild(clone);
01015
01016
01017                     PresentationAttributes currAttributes =
      InterpretPresentationAttributes(currObject, attributes, width, height, diagonal, gpr, styleSheets,
      gradients);
01018
01019
01020                     gpr.Save();
01021                     gpr.Translate(x, y);
01022
01023                     ((XmlElement)clone).SetAttribute("x", "0");
01024                     ((XmlElement)clone).SetAttribute("y", "0");
01025
01026                     if (clone.Attributes?["viewBox"] != null)
01027                     {
01028                         ((XmlElement)clone).SetAttribute("width",
      w.ToString(System.Globalization.CultureInfo.InvariantCulture));
01029                         ((XmlElement)clone).SetAttribute("height",
      h.ToString(System.Globalization.CultureInfo.InvariantCulture));
01030                     }
01031
01032                     InterpretSVGElement(clone, gpr, currAttributes, width, height, diagonal,
```

```
styleSheets, gradients, filters);
01033
01034                     gpr.Restore();
01035
01036                     if (currAttributes.NeedsRestore)
01037                     {
01038                         gpr.Restore();
01039                     }
01040                 }
01041             }
01042         }
01043
01044         private static bool ApplyClipPath(XmlNode currObject, Graphics gpr, double width, double
height, double diagonal, PresentationAttributes attributes, IEnumerable<Stylesheet> styleSheets,
Dictionary<string, (bool, Brush)> gradients)
01045         {
01046             string id = currObject.Attributes?["clip-path"]?.Value;
01047
01048             if (id != null && id.StartsWith("url(#"))
01049             {
01050                 id = id.Substring(5);
01051                 id = id.Substring(0, id.Length - 1);
01052
01053                 XmlNode element =
currObject.OwnerDocument.SelectSingleNode(string.Format("//*[@id='{0}']", id));
01054
01055                 if (element != null && element.ChildNodes.Count == 1 &&
element.ChildNodes[0].Name.Equals("path", StringComparison.OrdinalIgnoreCase))
01056                 {
01057                     bool hasParentClipPath = ApplyClipPath(element, gpr, width, height, diagonal,
attributes, styleSheets, gradients);
01058
01059                     Graphics pathGraphics = new Graphics();
01060                     InterpretPathObject(element.ChildNodes[0], pathGraphics, width, height, diagonal,
attributes, styleSheets, gradients);
01061
01062                     PathTransformerGraphicsContext ptgc = new PathTransformerGraphicsContext();
01063                     pathGraphics.CopyToIGraphicsContext(ptgc);
01064
01065                     if (!hasParentClipPath)
01066                     {
01067                         gpr.Save();
01068                     }
01069
01070                     gpr.SetClippingPath(ptgc.CurrentPath);
01071
01072                     return true;
01073                 }
01074                 else if (element != null && element.ChildNodes.Count == 1 &&
element.ChildNodes[0].Name.Equals("rect", StringComparison.OrdinalIgnoreCase))
01075                 {
01076                     bool hasParentClipPath = ApplyClipPath(element, gpr, width, height, diagonal,
attributes, styleSheets, gradients);
01077
01078                     Graphics pathGraphics = new Graphics();
01079                     InterpretRectObject(element.ChildNodes[0], pathGraphics, width, height, diagonal,
attributes, styleSheets, gradients);
01080
01081                     PathTransformerGraphicsContext ptgc = new PathTransformerGraphicsContext();
01082                     pathGraphics.CopyToIGraphicsContext(ptgc);
01083
01084                     if (!hasParentClipPath)
01085                     {
01086                         gpr.Save();
01087                     }
01088
01089                     gpr.SetClippingPath(ptgc.CurrentPath);
01090
01091                     return true;
01092                 }
01093
01094                 return false;
01095             }
01096             else
01097             {
01098                 return false;
01099             }
01100         }
01101
01102         private static void InterpretRectObject(XmlNode currObject, Graphics gpr, double width, double
height, double diagonal, PresentationAttributes attributes, IEnumerable<Stylesheet> styleSheets,
Dictionary<string, (bool, Brush)> gradients)
01103         {
01104             double x, y, w, h, rx, ry;
01105
01106             x = ParseLengthOrPercentage(currObject.Attributes?["x"]?.Value, width);
01107             y = ParseLengthOrPercentage(currObject.Attributes?["y"]?.Value, height);
```

```
01108             w = ParseLengthOrPercentage(currObject.Attributes?["width"]?.Value, width);
01109             h = ParseLengthOrPercentage(currObject.Attributes?["height"]?.Value, height);
01110             rx = ParseLengthOrPercentage(currObject.Attributes?["rx"]?.Value, width, double.NaN);
01111             ry = ParseLengthOrPercentage(currObject.Attributes?["ry"]?.Value, height, double.NaN);
01112 
01113             if (w > 0 && h > 0)
01114             {
01115                 if (double.IsNaN(rx) && !double.IsNaN(ry))
01116                 {
01117                     rx = ry;
01118                 }
01119                 else if (!double.IsNaN(rx) && double.IsNaN(ry))
01120                 {
01121                     ry = rx;
01122                 }
01123 
01124                 if (double.IsNaN(rx))
01125                 {
01126                     rx = 0;
01127                 }
01128 
01129                 if (double.IsNaN(ry))
01130                 {
01131                     ry = 0;
01132                 }
01133 
01134                 rx = Math.Min(rx, w / 2);
01135                 ry = Math.Min(ry, h / 2);
01136 
01137                 GraphicsPath path = new GraphicsPath();
01138 
01139                 path.MoveTo(x + rx, y);
01140                 path.LineTo(x + w - rx, y);
01141 
01142                 if (rx > 0 && ry > 0)
01143                 {
01144                     path.EllipticalArc(rx, ry, 0, false, true, new Point(x + w, y + ry));
01145                 }
01146 
01147                 path.LineTo(x + w, y + h - ry);
01148 
01149                 if (rx > 0 && ry > 0)
01150                 {
01151                     path.EllipticalArc(rx, ry, 0, false, true, new Point(x + w - rx, y + h));
01152                 }
01153 
01154                 path.LineTo(x + rx, y + h);
01155 
01156                 if (rx > 0 && ry > 0)
01157                 {
01158                     path.EllipticalArc(rx, ry, 0, false, true, new Point(x, y + h - ry));
01159                 }
01160 
01161                 path.LineTo(x, y + ry);
01162 
01163                 if (rx > 0 && ry > 0)
01164                 {
01165                     path.EllipticalArc(rx, ry, 0, false, true, new Point(x + rx, y));
01166                 }
01167 
01168                 path.Close();
01169 
01170                 PresentationAttributes currAttributes = InterpretPresentationAttributes(currObject,
      attributes, width, height, diagonal, gpr, styleSheets, gradients);
01171 
01172                 bool hadClippingPath = ApplyClipPath(currObject, gpr, width, height, diagonal,
      attributes, styleSheets, gradients);
01173 
01174                 string tag = currObject.Attributes?["id"]?.Value;
01175 
01176                 if (currAttributes.StrokeFirst)
01177                 {
01178                     if (currAttributes.Stroke != null)
01179                     {
01180                         Brush strokeColour =
      currAttributes.Stroke.MultiplyOpacity(currAttributes.Opacity * currAttributes.StrokeOpacity);
01181 
01182                         GraphicsPath oldPath = path;
01183                         double[,] transform = null;
01184 
01185                         if (currAttributes.StrokeGradientNeedsTransform)
01186                         {
01187                             (strokeColour, transform) = TransformBrush(strokeColour,
      path.GetBounds());
01188                         }
01189 
01190                         if (transform != null)
```

```
01191                         {
01192                             gpr.Save();
01193                             gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
01194                             path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
01195                         }
01196 
01197                         gpr.StrokePath(path, strokeColour, currAttributes.StrokeThickness,
      currAttributes.LineCap, currAttributes.LineJoin, currAttributes.LineDash, tag:  tag);
01198 
01199                         if (transform != null)
01200                         {
01201                             gpr.Restore();
01202                             path = oldPath;
01203                         }
01204                     }
01205 
01206                     if (currAttributes.Fill != null)
01207                     {
01208                         Brush fillColour = currAttributes.Fill.MultiplyOpacity(currAttributes.Opacity
      * currAttributes.FillOpacity);
01209 
01210                         GraphicsPath oldPath = path;
01211                         double[,] transform = null;
01212 
01213                         if (currAttributes.FillGradientNeedsTransform)
01214                         {
01215                             (fillColour, transform) = TransformBrush(fillColour, path.GetBounds());
01216                         }
01217 
01218                         if (transform != null)
01219                         {
01220                             gpr.Save();
01221                             gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
01222                             path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
01223                         }
01224 
01225                         gpr.FillPath(path, fillColour, tag:  tag);
01226 
01227                         if (transform != null)
01228                         {
01229                             gpr.Restore();
01230                             path = oldPath;
01231                         }
01232                     }
01233                 }
01234                 else
01235                 {
01236                     if (currAttributes.Fill != null)
01237                     {
01238                         Brush fillColour = currAttributes.Fill.MultiplyOpacity(currAttributes.Opacity
      * currAttributes.FillOpacity);
01239 
01240                         GraphicsPath oldPath = path;
01241                         double[,] transform = null;
01242 
01243                         if (currAttributes.FillGradientNeedsTransform)
01244                         {
01245                             (fillColour, transform) = TransformBrush(fillColour, path.GetBounds());
01246                         }
01247 
01248                         if (transform != null)
01249                         {
01250                             gpr.Save();
01251                             gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
01252                             path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
01253                         }
01254 
01255                         gpr.FillPath(path, fillColour, tag:  tag);
01256 
01257                         if (transform != null)
01258                         {
01259                             gpr.Restore();
01260                             path = oldPath;
01261                         }
01262                     }
01263 
01264                     if (currAttributes.Stroke != null)
01265                     {
01266                         Brush strokeColour =
      currAttributes.Stroke.MultiplyOpacity(currAttributes.Opacity * currAttributes.StrokeOpacity);
01267 
01268                         GraphicsPath oldPath = path;
01269                         double[,] transform = null;
01270 
```

```
01271                         if (currAttributes.StrokeGradientNeedsTransform)
01272                         {
01273                             (strokeColour, transform) = TransformBrush(strokeColour,
      path.GetBounds());
01274                         }
01275
01276                         if (transform != null)
01277                         {
01278                             gpr.Save();
01279                             gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
01280                             path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
01281                         }
01282
01283                         gpr.StrokePath(path, strokeColour, currAttributes.StrokeThickness,
      currAttributes.LineCap, currAttributes.LineJoin, currAttributes.LineDash, tag:  tag);
01284
01285                         if (transform != null)
01286                         {
01287                             gpr.Restore();
01288                             path = oldPath;
01289                         }
01290                     }
01291                 }
01292
01293                 if (hadClippingPath)
01294                 {
01295                     gpr.Restore();
01296                 }
01297
01298                 if (currAttributes.NeedsRestore)
01299                 {
01300                     gpr.Restore();
01301                 }
01302             }
01303         }
01304
01305         private static void InterpretPolyLineObject(XmlNode currObject, bool isPolygon, Graphics gpr,
      double width, double height, double diagonal, PresentationAttributes attributes,
      IEnumerable<Stylesheet> styleSheets, Dictionary<string, (bool, Brush)> gradients)
01306         {
01307             string points = currObject.Attributes?["points"]?.Value;
01308
01309             if (points != null)
01310             {
01311                 double[] coordinates = ParseListOfDoubles(points);
01312
01313                 GraphicsPath path = new GraphicsPath();
01314
01315                 for (int i = 0; i < coordinates.Length; i += 2)
01316                 {
01317                     path.LineTo(coordinates[i], coordinates[i + 1]);
01318                 }
01319
01320                 if (isPolygon)
01321                 {
01322                     path.Close();
01323                 }
01324
01325                 PresentationAttributes currAttributes = InterpretPresentationAttributes(currObject,
      attributes, width, height, diagonal, gpr, styleSheets, gradients);
01326
01327                 bool hadClippingPath = ApplyClipPath(currObject, gpr, width, height, diagonal,
      attributes, styleSheets, gradients);
01328
01329                 string tag = currObject.Attributes?["id"]?.Value;
01330
01331                 if (currAttributes.StrokeFirst)
01332                 {
01333                     if (currAttributes.Stroke != null)
01334                     {
01335                         Brush strokeColour =
      currAttributes.Stroke.MultiplyOpacity(currAttributes.Opacity * currAttributes.StrokeOpacity);
01336
01337                         GraphicsPath oldPath = path;
01338                         double[,] transform = null;
01339
01340                         if (currAttributes.StrokeGradientNeedsTransform)
01341                         {
01342                             (strokeColour, transform) = TransformBrush(strokeColour,
      path.GetBounds());
01343                         }
01344
01345                         if (transform != null)
01346                         {
01347                             gpr.Save();
01348                             gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
```

```
    transform[1, 1], transform[0, 2], transform[1, 2]);
01349                           path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
01350                       }
01351
01352                       gpr.StrokePath(path, strokeColour, currAttributes.StrokeThickness,
    currAttributes.LineCap, currAttributes.LineJoin, currAttributes.LineDash, tag:  tag);
01353
01354                       if (transform != null)
01355                       {
01356                           gpr.Restore();
01357                           path = oldPath;
01358                       }
01359                   }
01360
01361                   if (currAttributes.Fill != null)
01362                   {
01363                       Brush fillColour = currAttributes.Fill.MultiplyOpacity(currAttributes.Opacity
    * currAttributes.FillOpacity);
01364
01365                       GraphicsPath oldPath = path;
01366                       double[,] transform = null;
01367
01368                       if (currAttributes.FillGradientNeedsTransform)
01369                       {
01370                           (fillColour, transform) = TransformBrush(fillColour, path.GetBounds());
01371                       }
01372
01373                       if (transform != null)
01374                       {
01375                           gpr.Save();
01376                           gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
    transform[1, 1], transform[0, 2], transform[1, 2]);
01377                           path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
01378                       }
01379
01380                       gpr.FillPath(path, fillColour, tag:  tag);
01381
01382                       if (transform != null)
01383                       {
01384                           gpr.Restore();
01385                           path = oldPath;
01386                       }
01387                   }
01388               }
01389               else
01390               {
01391                   if (currAttributes.Fill != null)
01392                   {
01393                       Brush fillColour = currAttributes.Fill.MultiplyOpacity(currAttributes.Opacity
    * currAttributes.FillOpacity);
01394
01395                       GraphicsPath oldPath = path;
01396                       double[,] transform = null;
01397
01398                       if (currAttributes.FillGradientNeedsTransform)
01399                       {
01400                           (fillColour, transform) = TransformBrush(fillColour, path.GetBounds());
01401                       }
01402
01403                       if (transform != null)
01404                       {
01405                           gpr.Save();
01406                           gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
    transform[1, 1], transform[0, 2], transform[1, 2]);
01407                           path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
01408                       }
01409
01410                       gpr.FillPath(path, fillColour, tag:  tag);
01411
01412                       if (transform != null)
01413                       {
01414                           gpr.Restore();
01415                           path = oldPath;
01416                       }
01417                   }
01418
01419                   if (currAttributes.Stroke != null)
01420                   {
01421                       Brush strokeColour =
    currAttributes.Stroke.MultiplyOpacity(currAttributes.Opacity * currAttributes.StrokeOpacity);
01422
01423                       GraphicsPath oldPath = path;
01424                       double[,] transform = null;
01425
01426                       if (currAttributes.StrokeGradientNeedsTransform)
01427                       {
01428                           (strokeColour, transform) = TransformBrush(strokeColour,
```

```
path.GetBounds());
01429                         }
01430
01431                         if (transform != null)
01432                         {
01433                             gpr.Save();
01434                             gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
transform[1, 1], transform[0, 2], transform[1, 2]);
01435                             path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
01436                         }
01437
01438                         gpr.StrokePath(path, strokeColour, currAttributes.StrokeThickness,
currAttributes.LineCap, currAttributes.LineJoin, currAttributes.LineDash, tag:  tag);
01439
01440                         if (transform != null)
01441                         {
01442                             gpr.Restore();
01443                             path = oldPath;
01444                         }
01445                     }
01446                 }
01447
01448                 if (hadClippingPath)
01449                 {
01450                     gpr.Restore();
01451                 }
01452
01453                 if (currAttributes.NeedsRestore)
01454                 {
01455                     gpr.Restore();
01456                 }
01457             }
01458         }
01459
01460         private static void InterpretPathObject(XmlNode currObject, Graphics gpr, double width, double
height, double diagonal, PresentationAttributes attributes, IEnumerable<Stylesheet> styleSheets,
Dictionary<string, (bool, Brush)> gradients)
01461         {
01462             string d = currObject.Attributes?["d"]?.Value;
01463
01464             if (d != null)
01465             {
01466                 List<string> pathData = TokenisePathData(d);
01467
01468                 GraphicsPath path = new GraphicsPath();
01469
01470                 Point lastPoint = new Point();
01471                 Point?  figureStartPoint = null;
01472
01473                 char lastCommand = '\0';
01474                 Point lastCtrlPoint = new Point();
01475
01476                 for (int i = 0; i < pathData.Count; i++)
01477                 {
01478                     Point delta = new Point();
01479
01480                     bool isAbsolute = char.IsUpper(pathData[i][0]);
01481
01482                     if (!isAbsolute)
01483                     {
01484                         delta = lastPoint;
01485                     }
01486
01487                     switch (pathData[i][0])
01488                     {
01489                         case 'M':
01490                         case 'm':
01491                             lastPoint = new Point(delta.X + double.Parse(pathData[i + 1],
System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
System.Globalization.CultureInfo.InvariantCulture));
01492                             path.MoveTo(lastPoint);
01493                             figureStartPoint = lastPoint;
01494                             i += 2;
01495                             lastCommand = 'M';
01496                             while (i < pathData.Count - 1 && !char.IsLetter(pathData[i + 1][0]))
01497                             {
01498                                 if (!isAbsolute)
01499                                 {
01500                                     delta = lastPoint;
01501                                 }
01502                                 else
01503                                 {
01504                                     delta = new Point();
01505                                 }
01506
01507                                 lastPoint = new Point(delta.X + double.Parse(pathData[i + 1],
System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
```

```
      System.Globalization.CultureInfo.InvariantCulture));
01508                                 path.LineTo(lastPoint);
01509
01510                                 i += 2;
01511                                 lastCommand = 'L';
01512                             }
01513                             break;
01514                         case 'L':
01515                         case 'l':
01516                             lastPoint = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01517                             path.LineTo(lastPoint);
01518                             if (figureStartPoint == null)
01519                             {
01520                                 figureStartPoint = lastPoint;
01521                             }
01522                             i += 2;
01523                             lastCommand = 'L';
01524                             while (i < pathData.Count - 1 && !char.IsLetter(pathData[i + 1][0]))
01525                             {
01526                                 if (!isAbsolute)
01527                                 {
01528                                     delta = lastPoint;
01529                                 }
01530                                 else
01531                                 {
01532                                     delta = new Point();
01533                                 }
01534
01535                                 lastPoint = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01536                                 path.LineTo(lastPoint);
01537
01538                                 i += 2;
01539                             }
01540                             break;
01541                         case 'H':
01542                         case 'h':
01543                             lastPoint = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), lastPoint.Y);
01544                             path.LineTo(lastPoint);
01545                             if (figureStartPoint == null)
01546                             {
01547                                 figureStartPoint = lastPoint;
01548                             }
01549                             i++;
01550                             lastCommand = 'L';
01551                             while (i < pathData.Count - 1 && !char.IsLetter(pathData[i + 1][0]))
01552                             {
01553                                 if (!isAbsolute)
01554                                 {
01555                                     delta = lastPoint;
01556                                 }
01557                                 else
01558                                 {
01559                                     delta = new Point();
01560                                 }
01561
01562                                 lastPoint = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), lastPoint.Y);
01563                                 path.LineTo(lastPoint);
01564
01565                                 i++;
01566                             }
01567                             break;
01568                         case 'V':
01569                         case 'v':
01570                             lastPoint = new Point(lastPoint.X, delta.Y + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture));
01571                             path.LineTo(lastPoint);
01572                             if (figureStartPoint == null)
01573                             {
01574                                 figureStartPoint = lastPoint;
01575                             }
01576                             i++;
01577                             lastCommand = 'L';
01578                             while (i < pathData.Count - 1 && !char.IsLetter(pathData[i + 1][0]))
01579                             {
01580                                 if (!isAbsolute)
01581                                 {
01582                                     delta = lastPoint;
01583                                 }
01584                                 else
01585                                 {
01586                                     delta = new Point();
```

```
01587                                 }
01588
01589                                 lastPoint = new Point(lastPoint.X, delta.Y + double.Parse(pathData[i +
      1], System.Globalization.CultureInfo.InvariantCulture));
01590                                 path.LineTo(lastPoint);
01591
01592                                 i++;
01593                             }
01594                             break;
01595                         case 'C':
01596                         case 'c':
01597                             {
01598                                 Point ctrlPoint1 = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01599                                 i += 2;
01600
01601                                 Point ctrlPoint2 = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01602                                 i += 2;
01603
01604                                 lastPoint = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01605                                 i += 2;
01606
01607                                 if (figureStartPoint == null)
01608                                 {
01609                                     figureStartPoint = lastPoint;
01610                                 }
01611
01612                                 path.CubicBezierTo(ctrlPoint1, ctrlPoint2, lastPoint);
01613
01614                                 lastCtrlPoint = ctrlPoint2;
01615                                 lastCommand = 'C';
01616
01617                                 while (i < pathData.Count - 1 && !char.IsLetter(pathData[i + 1][0]))
01618                                 {
01619                                     if (!isAbsolute)
01620                                     {
01621                                         delta = lastPoint;
01622                                     }
01623                                     else
01624                                     {
01625                                         delta = new Point();
01626                                     }
01627
01628                                     ctrlPoint1 = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01629                                     i += 2;
01630
01631                                     ctrlPoint2 = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01632                                     i += 2;
01633
01634                                     lastPoint = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01635                                     i += 2;
01636
01637                                     path.CubicBezierTo(ctrlPoint1, ctrlPoint2, lastPoint);
01638                                     lastCtrlPoint = ctrlPoint2;
01639                                 }
01640                             }
01641                             break;
01642                         case 'S':
01643                         case 's':
01644                             {
01645                                 Point ctrlPoint1;
01646
01647                                 if (lastCommand == 'C')
01648                                 {
01649                                     ctrlPoint1 = new Point(2 * lastPoint.X - lastCtrlPoint.X, 2 *
      lastPoint.Y - lastCtrlPoint.Y);
01650                                 }
01651                                 else
01652                                 {
01653                                     ctrlPoint1 = lastPoint;
01654                                 }
01655
01656                                 Point ctrlPoint2 = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01657                                 i += 2;
```

```
01658
01659                                   lastPoint = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01660                                   i += 2;
01661
01662                                   if (figureStartPoint == null)
01663                                   {
01664                                       figureStartPoint = lastPoint;
01665                                   }
01666
01667                                   path.CubicBezierTo(ctrlPoint1, ctrlPoint2, lastPoint);
01668
01669                                   lastCtrlPoint = ctrlPoint2;
01670                                   lastCommand = 'C';
01671
01672                                   while (i < pathData.Count - 1 && !char.IsLetter(pathData[i + 1][0]))
01673                                   {
01674                                       if (!isAbsolute)
01675                                       {
01676                                           delta = lastPoint;
01677                                       }
01678                                       else
01679                                       {
01680                                           delta = new Point();
01681                                       }
01682
01683                                       ctrlPoint1 = new Point(2 * lastPoint.X - lastCtrlPoint.X, 2 *
      lastPoint.Y - lastCtrlPoint.Y);
01684
01685                                       ctrlPoint2 = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01686                                       i += 2;
01687
01688                                       lastPoint = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01689                                       i += 2;
01690
01691                                       path.CubicBezierTo(ctrlPoint1, ctrlPoint2, lastPoint);
01692
01693                                       lastCtrlPoint = ctrlPoint2;
01694                                   }
01695                               }
01696                               break;
01697                           case 'Q':
01698                           case 'q':
01699                               {
01700                                   Point ctrlPoint = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01701                                   i += 2;
01702
01703                                   Point actualCP1 = new Point(lastPoint.X + 2 * (ctrlPoint.X -
      lastPoint.X) / 3, lastPoint.Y + 2 * (ctrlPoint.Y - lastPoint.Y) / 3);
01704
01705                                   lastPoint = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01706                                   i += 2;
01707
01708                                   Point actualCP2 = new Point(lastPoint.X + 2 * (ctrlPoint.X -
      lastPoint.X) / 3, lastPoint.Y + 2 * (ctrlPoint.Y - lastPoint.Y) / 3);
01709
01710                                   if (figureStartPoint == null)
01711                                   {
01712                                       figureStartPoint = lastPoint;
01713                                   }
01714
01715                                   path.CubicBezierTo(actualCP1, actualCP2, lastPoint);
01716
01717                                   lastCtrlPoint = ctrlPoint;
01718                                   lastCommand = 'Q';
01719
01720                                   while (i < pathData.Count - 1 && !char.IsLetter(pathData[i + 1][0]))
01721                                   {
01722                                       if (!isAbsolute)
01723                                       {
01724                                           delta = lastPoint;
01725                                       }
01726                                       else
01727                                       {
01728                                           delta = new Point();
01729                                       }
01730
01731                                       ctrlPoint = new Point(delta.X + double.Parse(pathData[i + 1],
```

```
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01732                                 i += 2;
01733
01734                                 actualCP1 = new Point(lastPoint.X + 2 * (ctrlPoint.X -
      lastPoint.X) / 3, lastPoint.Y + 2 * (ctrlPoint.Y - lastPoint.Y) / 3);
01735
01736                                 lastPoint = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01737                                 i += 2;
01738
01739                                 actualCP2 = new Point(lastPoint.X + 2 * (ctrlPoint.X -
      lastPoint.X) / 3, lastPoint.Y + 2 * (ctrlPoint.Y - lastPoint.Y) / 3);
01740
01741                                 path.CubicBezierTo(actualCP1, actualCP2, lastPoint);
01742                                 lastCtrlPoint = ctrlPoint;
01743                             }
01744
01745
01746                         }
01747                         break;
01748                     case 'T':
01749                     case 't':
01750                         {
01751                             Point ctrlPoint;
01752
01753                             if (lastCommand == 'Q')
01754                             {
01755                                 ctrlPoint = new Point(2 * lastPoint.X - lastCtrlPoint.X, 2 *
      lastPoint.Y - lastCtrlPoint.Y);
01756                             }
01757                             else
01758                             {
01759                                 ctrlPoint = lastPoint;
01760                             }
01761
01762                             Point actualCP1 = new Point(lastPoint.X + 2 * (ctrlPoint.X -
      lastPoint.X) / 3, lastPoint.Y + 2 * (ctrlPoint.Y - lastPoint.Y) / 3);
01763
01764                             lastPoint = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01765                             i += 2;
01766
01767                             Point actualCP2 = new Point(lastPoint.X + 2 * (ctrlPoint.X -
      lastPoint.X) / 3, lastPoint.Y + 2 * (ctrlPoint.Y - lastPoint.Y) / 3);
01768
01769                             if (figureStartPoint == null)
01770                             {
01771                                 figureStartPoint = lastPoint;
01772                             }
01773
01774                             path.CubicBezierTo(actualCP1, actualCP2, lastPoint);
01775                             lastCtrlPoint = ctrlPoint;
01776                             lastCommand = 'Q';
01777
01778                             while (i < pathData.Count - 1 && !char.IsLetter(pathData[i + 1][0]))
01779                             {
01780                                 if (!isAbsolute)
01781                                 {
01782                                     delta = lastPoint;
01783                                 }
01784                                 else
01785                                 {
01786                                     delta = new Point();
01787                                 }
01788
01789                                 ctrlPoint = new Point(2 * lastPoint.X - lastCtrlPoint.X, 2 *
      lastPoint.Y - lastCtrlPoint.Y);
01790
01791                                 actualCP1 = new Point(lastPoint.X + 2 * (ctrlPoint.X -
      lastPoint.X) / 3, lastPoint.Y + 2 * (ctrlPoint.Y - lastPoint.Y) / 3);
01792
01793                                 lastPoint = new Point(delta.X + double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01794                                 i += 2;
01795
01796                                 actualCP2 = new Point(lastPoint.X + 2 * (ctrlPoint.X -
      lastPoint.X) / 3, lastPoint.Y + 2 * (ctrlPoint.Y - lastPoint.Y) / 3);
01797
01798                                 path.CubicBezierTo(actualCP1, actualCP2, lastPoint);
01799
01800                                 lastCtrlPoint = ctrlPoint;
01801                             }
01802                         }
```

```
01803                           break;
01804                       case 'A':
01805                       case 'a':
01806                           {
01807                               Point startPoint = lastPoint;
01808 
01809                               if (figureStartPoint == null)
01810                               {
01811                                   figureStartPoint = lastPoint;
01812                               }
01813 
01814                               Point radii = new Point(double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01815                               double angle = double.Parse(pathData[i + 3],
      System.Globalization.CultureInfo.InvariantCulture) * Math.PI / 180;
01816                               bool largeArcFlag = pathData[i + 4][0] == '1';
01817                               bool sweepFlag = pathData[i + 5][0] == '1';
01818 
01819                               lastPoint = new Point(delta.X + double.Parse(pathData[i + 6],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 7],
      System.Globalization.CultureInfo.InvariantCulture));
01820                               i += 7;
01821 
01822                               path.EllipticalArc(radii.X, radii.Y, angle, largeArcFlag, sweepFlag,
      lastPoint);
01823 
01824                               while (i < pathData.Count - 1 && !char.IsLetter(pathData[i + 1][0]))
01825                               {
01826                                   if (!isAbsolute)
01827                                   {
01828                                       delta = lastPoint;
01829                                   }
01830                                   else
01831                                   {
01832                                       delta = new Point();
01833                                   }
01834 
01835                                   startPoint = lastPoint;
01836                                   radii = new Point(double.Parse(pathData[i + 1],
      System.Globalization.CultureInfo.InvariantCulture), double.Parse(pathData[i + 2],
      System.Globalization.CultureInfo.InvariantCulture));
01837                                   angle = double.Parse(pathData[i + 3],
      System.Globalization.CultureInfo.InvariantCulture) * Math.PI / 180;
01838                                   largeArcFlag = pathData[i + 4][0] == '1';
01839                                   sweepFlag = pathData[i + 5][0] == '1';
01840 
01841                                   lastPoint = new Point(delta.X + double.Parse(pathData[i + 6],
      System.Globalization.CultureInfo.InvariantCulture), delta.Y + double.Parse(pathData[i + 7],
      System.Globalization.CultureInfo.InvariantCulture));
01842                                   i += 7;
01843 
01844                                   path.EllipticalArc(radii.X, radii.Y, angle, largeArcFlag,
      sweepFlag, lastPoint);
01845                               }
01846                           }
01847 
01848                           break;
01849                       case 'Z':
01850                       case 'z':
01851                           path.Close();
01852                           lastPoint = figureStartPoint.Value;
01853                           figureStartPoint = null;
01854                           lastCommand = 'Z';
01855                           break;
01856                   }
01857               }
01858 
01859               PresentationAttributes currAttributes = InterpretPresentationAttributes(currObject,
      attributes, width, height, diagonal, gpr, styleSheets, gradients);
01860 
01861               bool hadClippingPath = ApplyClipPath(currObject, gpr, width, height, diagonal,
      attributes, styleSheets, gradients);
01862 
01863               string tag = currObject.Attributes?["id"]?.Value;
01864 
01865               if (currAttributes.StrokeFirst)
01866               {
01867                   if (currAttributes.Stroke != null)
01868                   {
01869                       Brush strokeColour =
      currAttributes.Stroke.MultiplyOpacity(currAttributes.Opacity * currAttributes.StrokeOpacity);
01870 
01871                       GraphicsPath oldPath = path;
01872                       double[,] transform = null;
01873 
01874                       if (currAttributes.StrokeGradientNeedsTransform)
```

```
01875                         {
01876                             (strokeColour, transform) = TransformBrush(strokeColour,
      path.GetBounds());
01877                         }
01878 
01879                         if (transform != null)
01880                         {
01881                             gpr.Save();
01882                             gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
01883                             path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
01884                         }
01885 
01886                         gpr.StrokePath(path, strokeColour, currAttributes.StrokeThickness,
      currAttributes.LineCap, currAttributes.LineJoin, currAttributes.LineDash, tag:  tag);
01887 
01888                         if (transform != null)
01889                         {
01890                             gpr.Restore();
01891                             path = oldPath;
01892                         }
01893                     }
01894 
01895                     if (currAttributes.Fill != null)
01896                     {
01897                         Brush fillColour = currAttributes.Fill.MultiplyOpacity(currAttributes.Opacity
      * currAttributes.FillOpacity);
01898 
01899                         GraphicsPath oldPath = path;
01900                         double[,] transform = null;
01901 
01902                         if (currAttributes.FillGradientNeedsTransform)
01903                         {
01904                             (fillColour, transform) = TransformBrush(fillColour, path.GetBounds());
01905                         }
01906 
01907                         if (transform != null)
01908                         {
01909                             gpr.Save();
01910                             gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
01911                             path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
01912                         }
01913 
01914                         gpr.FillPath(path, fillColour, tag:  tag);
01915 
01916                         if (transform != null)
01917                         {
01918                             gpr.Restore();
01919                             path = oldPath;
01920                         }
01921                     }
01922                 }
01923                 else
01924                 {
01925                     if (currAttributes.Fill != null)
01926                     {
01927                         Brush fillColour = currAttributes.Fill.MultiplyOpacity(currAttributes.Opacity
      * currAttributes.FillOpacity);
01928 
01929                         GraphicsPath oldPath = path;
01930                         double[,] transform = null;
01931 
01932                         if (currAttributes.FillGradientNeedsTransform)
01933                         {
01934                             (fillColour, transform) = TransformBrush(fillColour, path.GetBounds());
01935                         }
01936 
01937                         if (transform != null)
01938                         {
01939                             gpr.Save();
01940                             gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
01941                             path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
01942                         }
01943 
01944                         gpr.FillPath(path, fillColour, tag:  tag);
01945 
01946                         if (transform != null)
01947                         {
01948                             gpr.Restore();
01949                             path = oldPath;
01950                         }
01951                     }
01952 
01953                     if (currAttributes.Stroke != null)
01954                     {
```

```
01955                       Brush strokeColour =
      currAttributes.Stroke.MultiplyOpacity(currAttributes.Opacity * currAttributes.StrokeOpacity);
01956                       GraphicsPath oldPath = path;
01957                       double[,] transform = null;
01958
01959                       if (currAttributes.StrokeGradientNeedsTransform)
01960                       {
01961                           (strokeColour, transform) = TransformBrush(strokeColour,
      path.GetBounds());
01962                       }
01963
01964                       if (transform != null)
01965                       {
01966                           gpr.Save();
01967                           gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
01968                           path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
01969                       }
01970
01971                       gpr.StrokePath(path, strokeColour, currAttributes.StrokeThickness,
      currAttributes.LineCap, currAttributes.LineJoin, currAttributes.LineDash, tag:  tag);
01972
01973                       if (transform != null)
01974                       {
01975                           gpr.Restore();
01976                           path = oldPath;
01977                       }
01978                   }
01979               }
01980
01981               if (hadClippingPath)
01982               {
01983                   gpr.Restore();
01984               }
01985
01986               if (currAttributes.NeedsRestore)
01987               {
01988                   gpr.Restore();
01989               }
01990           }
01991
01992       }
01993
01994       private static List<string> TokenisePathData(string d)
01995       {
01996           List<string> tbr = new List<string>();
01997
01998           string currToken = "";
01999
02000           for (int i = 0; i < d.Length; i++)
02001           {
02002               char c = d[i];
02003
02004               if (c >= '0' && c <= '9' || c == 'e' || c == 'E')
02005               {
02006                   currToken += c;
02007               }
02008               else if (c == '.')
02009               {
02010                   if (!currToken.Contains('.'))
02011                   {
02012                       currToken += c;
02013                   }
02014                   else
02015                   {
02016                       if (!string.IsNullOrEmpty(currToken))
02017                       {
02018                           tbr.Add(currToken);
02019                       }
02020                       currToken = "" + c;
02021                   }
02022               }
02023               else if (c == '-' || c == '+')
02024               {
02025                   if (i > 0 && (d[i - 1] == 'e' || d[i - 1] == 'E'))
02026                   {
02027                       currToken += c;
02028                   }
02029                   else
02030                   {
02031                       if (!string.IsNullOrEmpty(currToken))
02032                       {
02033                           tbr.Add(currToken);
02034                       }
02035                       currToken = "" + c;
02036                   }
02037               }
```

```
02038                 else if (char.IsWhiteSpace(c) || c == ',')
02039                 {
02040                     if (!string.IsNullOrEmpty(currToken))
02041                     {
02042                         tbr.Add(currToken);
02043                     }
02044                     currToken = "";
02045                 }
02046                 else if (i < d.Length - 2 && (c == 'N' || c == 'n') && (d[i + 1] == 'a' || d[i + 1] ==
'A') && (d[i + 2] == 'N' || d[i + 2] == 'n'))
02047                 {
02048                     if (!string.IsNullOrEmpty(currToken))
02049                     {
02050                         tbr.Add(currToken);
02051                     }
02052                     tbr.Add("NaN");
02053                     currToken = "";
02054                     i += 2;
02055                 }
02056                 else if ("MmLlHhVvCcSsQqTtAaZz".Contains(c))
02057                 {
02058                     if (!string.IsNullOrEmpty(currToken))
02059                     {
02060                         tbr.Add(currToken);
02061                     }
02062                     tbr.Add(c.ToString());
02063                     currToken = "";
02064                 }
02065             }
02066
02067             if (!string.IsNullOrEmpty(currToken))
02068             {
02069                 tbr.Add(currToken);
02070             }
02071
02072             return tbr;
02073         }
02074
02075         private static void InterpretCircleObject(XmlNode circleObject, Graphics gpr, double width,
double height, double diagonal, PresentationAttributes attributes, IEnumerable<Stylesheet>
styleSheets, Dictionary<string, (bool, Brush)> gradients)
02076         {
02077             double cx, cy, r;
02078
02079             cx = ParseLengthOrPercentage(circleObject.Attributes?["cx"]?.Value, width);
02080             cy = ParseLengthOrPercentage(circleObject.Attributes?["cy"]?.Value, height);
02081             r = ParseLengthOrPercentage(circleObject.Attributes?["r"]?.Value, diagonal);
02082
02083             string tag = circleObject.Attributes?["id"]?.Value;
02084
02085             PresentationAttributes circleAttributes = InterpretPresentationAttributes(circleObject,
attributes, width, height, diagonal, gpr, styleSheets, gradients);
02086
02087             bool hadClippingPath = ApplyClipPath(circleObject, gpr, width, height, diagonal,
attributes, styleSheets, gradients);
02088
02089             if (circleAttributes.StrokeFirst)
02090             {
02091                 if (circleAttributes.Stroke != null)
02092                 {
02093                     Brush strokeColour =
circleAttributes.Stroke.MultiplyOpacity(circleAttributes.Opacity * circleAttributes.StrokeOpacity);
02094
02095                     GraphicsPath path = new GraphicsPath().Arc(cx, cy, r, 0, 2 * Math.PI).Close();
02096                     GraphicsPath oldPath = path;
02097                     double[,] transform = null;
02098
02099                     if (circleAttributes.StrokeGradientNeedsTransform)
02100                     {
02101                         (strokeColour, transform) = TransformBrush(strokeColour, new Rectangle(cx - r,
cy - r, r * 2, r * 2));
02102                     }
02103
02104                     if (transform != null)
02105                     {
02106                         gpr.Save();
02107                         gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1], transform[1,
1], transform[0, 2], transform[1, 2]);
02108                         path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
02109                     }
02110
02111                     gpr.StrokePath(path, strokeColour, circleAttributes.StrokeThickness,
circleAttributes.LineCap, circleAttributes.LineJoin, circleAttributes.LineDash, tag:  tag);
02112
02113                     if (transform != null)
02114                     {
02115                         gpr.Restore();
```

```
02116                         path = oldPath;
02117                     }
02118                 }
02119
02120                 if (circleAttributes.Fill != null)
02121                 {
02122                     Brush fillColour = circleAttributes.Fill.MultiplyOpacity(circleAttributes.Opacity
      * circleAttributes.FillOpacity);
02123
02124                     GraphicsPath path = new GraphicsPath().Arc(cx, cy, r, 0, 2 * Math.PI).Close();
02125                     GraphicsPath oldPath = path;
02126                     double[,] transform = null;
02127
02128                     if (circleAttributes.FillGradientNeedsTransform)
02129                     {
02130                         (fillColour, transform) = TransformBrush(fillColour, new Rectangle(cx - r, cy
      - r, r * 2, r * 2));
02131                     }
02132
02133                     if (transform != null)
02134                     {
02135                         gpr.Save();
02136                         gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1], transform[1,
      1], transform[0, 2], transform[1, 2]);
02137                         path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
02138                     }
02139
02140                     gpr.FillPath(path, fillColour, tag:  tag);
02141
02142                     if (transform != null)
02143                     {
02144                         gpr.Restore();
02145                         path = oldPath;
02146                     }
02147                 }
02148             }
02149             else
02150             {
02151                 if (circleAttributes.Fill != null)
02152                 {
02153                     Brush fillColour = circleAttributes.Fill.MultiplyOpacity(circleAttributes.Opacity
      * circleAttributes.FillOpacity);
02154                     GraphicsPath path = new GraphicsPath().Arc(cx, cy, r, 0, 2 * Math.PI).Close();
02155                     GraphicsPath oldPath = path;
02156                     double[,] transform = null;
02157
02158                     if (circleAttributes.FillGradientNeedsTransform)
02159                     {
02160                         (fillColour, transform) = TransformBrush(fillColour, new Rectangle(cx - r, cy
      - r, r * 2, r * 2));
02161                     }
02162
02163                     if (transform != null)
02164                     {
02165                         gpr.Save();
02166                         gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1], transform[1,
      1], transform[0, 2], transform[1, 2]);
02167                         path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
02168                     }
02169
02170                     gpr.FillPath(path, fillColour, tag:  tag);
02171
02172                     if (transform != null)
02173                     {
02174                         gpr.Restore();
02175                         path = oldPath;
02176                     }
02177                 }
02178
02179                 if (circleAttributes.Stroke != null)
02180                 {
02181                     Brush strokeColour =
      circleAttributes.Stroke.MultiplyOpacity(circleAttributes.Opacity * circleAttributes.StrokeOpacity);
02182                     GraphicsPath path = new GraphicsPath().Arc(cx, cy, r, 0, 2 * Math.PI).Close();
02183                     GraphicsPath oldPath = path;
02184                     double[,] transform = null;
02185
02186                     if (circleAttributes.StrokeGradientNeedsTransform)
02187                     {
02188                         (strokeColour, transform) = TransformBrush(strokeColour, new Rectangle(cx - r,
      cy - r, r * 2, r * 2));
02189                     }
02190
02191                     if (transform != null)
02192                     {
02193                         gpr.Save();
02194                         gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1], transform[1,
```

```
1], transform[0, 2], transform[1, 2]);
02195                       path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
02196                   }
02197
02198                   gpr.StrokePath(path, strokeColour, circleAttributes.StrokeThickness,
circleAttributes.LineCap, circleAttributes.LineJoin, circleAttributes.LineDash, tag:  tag);
02199
02200                   if (transform != null)
02201                   {
02202                       gpr.Restore();
02203                       path = oldPath;
02204                   }
02205               }
02206           }
02207
02208           if (hadClippingPath)
02209           {
02210               gpr.Restore();
02211           }
02212
02213           if (circleAttributes.NeedsRestore)
02214           {
02215               gpr.Restore();
02216           }
02217       }
02218
02219       private static void InterpretEllipseObject(XmlNode currObject, Graphics gpr, double width,
double height, double diagonal, PresentationAttributes attributes, IEnumerable<Stylesheet>
styleSheets, Dictionary<string, (bool, Brush)> gradients)
02220       {
02221           double cx, cy, rx, ry;
02222
02223           cx = ParseLengthOrPercentage(currObject.Attributes?["cx"]?.Value, width);
02224           cy = ParseLengthOrPercentage(currObject.Attributes?["cy"]?.Value, height);
02225           rx = ParseLengthOrPercentage(currObject.Attributes?["rx"]?.Value, width, double.NaN);
02226           ry = ParseLengthOrPercentage(currObject.Attributes?["ry"]?.Value, height, double.NaN);
02227
02228           string tag = currObject.Attributes?["id"]?.Value;
02229
02230           if (double.IsNaN(rx) && !double.IsNaN(ry))
02231           {
02232               rx = ry;
02233           }
02234           else if (!double.IsNaN(rx) && double.IsNaN(ry))
02235           {
02236               ry = rx;
02237           }
02238
02239           if (rx > 0 && ry > 0)
02240           {
02241
02242               PresentationAttributes currAttributes = InterpretPresentationAttributes(currObject,
attributes, width, height, diagonal, gpr, styleSheets, gradients);
02243
02244               bool hadClippingPath = ApplyClipPath(currObject, gpr, width, height, diagonal,
attributes, styleSheets, gradients);
02245
02246               double r = Math.Min(rx, ry);
02247
02248               gpr.Save();
02249               gpr.Translate(cx, cy);
02250               gpr.Scale(rx / r, ry / r);
02251
02252               if (currAttributes.StrokeFirst)
02253               {
02254                   if (currAttributes.Stroke != null)
02255                   {
02256                       Brush strokeColour =
currAttributes.Stroke.MultiplyOpacity(currAttributes.Opacity * currAttributes.StrokeOpacity);
02257
02258                       GraphicsPath path = new GraphicsPath().Arc(0, 0, r, 0, 2 * Math.PI).Close();
02259                       GraphicsPath oldPath = path;
02260                       double[,] transform = null;
02261
02262                       if (currAttributes.StrokeGradientNeedsTransform)
02263                       {
02264                           (strokeColour, transform) = TransformBrush(strokeColour, new Rectangle(0,
0, r * 2, r * 2));
02265                       }
02266
02267                       if (transform != null)
02268                       {
02269                           gpr.Save();
02270                           gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
transform[1, 1], transform[0, 2], transform[1, 2]);
02271                           path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
02272                       }
```

```
02273
02274                           gpr.StrokePath(path, strokeColour, currAttributes.StrokeThickness,
      currAttributes.LineCap, currAttributes.LineJoin, currAttributes.LineDash, tag:  tag);
02275
02276                           if (transform != null)
02277                           {
02278                               gpr.Restore();
02279                               path = oldPath;
02280                           }
02281                       }
02282
02283                       if (currAttributes.Fill != null)
02284                       {
02285                           Brush fillColour = currAttributes.Fill.MultiplyOpacity(currAttributes.Opacity
      * currAttributes.FillOpacity);
02286
02287                           GraphicsPath path = new GraphicsPath().Arc(0, 0, r, 0, 2 * Math.PI).Close();
02288                           GraphicsPath oldPath = path;
02289                           double[,] transform = null;
02290
02291                           if (currAttributes.FillGradientNeedsTransform)
02292                           {
02293                               (fillColour, transform) = TransformBrush(fillColour, new Rectangle(0, 0, r
      * 2, r * 2));
02294                           }
02295
02296                           if (transform != null)
02297                           {
02298                               gpr.Save();
02299                               gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
02300                               path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
02301                           }
02302
02303                           gpr.FillPath(path, fillColour, tag:  tag);
02304
02305                           if (transform != null)
02306                           {
02307                               gpr.Restore();
02308                               path = oldPath;
02309                           }
02310                       }
02311                   }
02312                   else
02313                   {
02314                       if (currAttributes.Fill != null)
02315                       {
02316                           Brush fillColour = currAttributes.Fill.MultiplyOpacity(currAttributes.Opacity
      * currAttributes.FillOpacity);
02317
02318                           GraphicsPath path = new GraphicsPath().Arc(0, 0, r, 0, 2 * Math.PI).Close();
02319                           GraphicsPath oldPath = path;
02320                           double[,] transform = null;
02321
02322                           if (currAttributes.FillGradientNeedsTransform)
02323                           {
02324                               (fillColour, transform) = TransformBrush(fillColour, new Rectangle(0, 0, r
      * 2, r * 2));
02325                           }
02326
02327                           if (transform != null)
02328                           {
02329                               gpr.Save();
02330                               gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
02331                               path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
02332                           }
02333
02334                           gpr.FillPath(path, fillColour, tag:  tag);
02335
02336                           if (transform != null)
02337                           {
02338                               gpr.Restore();
02339                               path = oldPath;
02340                           }
02341                       }
02342
02343                       if (currAttributes.Stroke != null)
02344                       {
02345                           Brush strokeColour =
      currAttributes.Stroke.MultiplyOpacity(currAttributes.Opacity * currAttributes.StrokeOpacity);
02346                           GraphicsPath path = new GraphicsPath().Arc(0, 0, r, 0, 2 * Math.PI).Close();
02347                           GraphicsPath oldPath = path;
02348                           double[,] transform = null;
02349
02350                           if (currAttributes.StrokeGradientNeedsTransform)
02351                           {
```

```
02352                           (strokeColour, transform) = TransformBrush(strokeColour, new Rectangle(0,
      0, r * 2, r * 2));
02353                       }
02354
02355                       if (transform != null)
02356                       {
02357                           gpr.Save();
02358                           gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1],
      transform[1, 1], transform[0, 2], transform[1, 2]);
02359                           path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
02360                       }
02361
02362                       gpr.StrokePath(path, strokeColour, currAttributes.StrokeThickness,
      currAttributes.LineCap, currAttributes.LineJoin, currAttributes.LineDash, tag:  tag);
02363
02364                       if (transform != null)
02365                       {
02366                           gpr.Restore();
02367                           path = oldPath;
02368                       }
02369                   }
02370               }
02371
02372               gpr.Restore();
02373
02374               if (hadClippingPath)
02375               {
02376                   gpr.Restore();
02377               }
02378
02379               if (currAttributes.NeedsRestore)
02380               {
02381                   gpr.Restore();
02382               }
02383           }
02384       }
02385
02386       private static void InterpretLineObject(XmlNode lineObject, Graphics gpr, double width, double
      height, double diagonal, PresentationAttributes attributes, IEnumerable<Stylesheet> styleSheets,
      Dictionary<string, (bool, Brush)> gradients)
02387       {
02388           double x1, x2, y1, y2;
02389
02390           x1 = ParseLengthOrPercentage(lineObject.Attributes?["x1"]?.Value, width);
02391           y1 = ParseLengthOrPercentage(lineObject.Attributes?["y1"]?.Value, height);
02392           x2 = ParseLengthOrPercentage(lineObject.Attributes?["x2"]?.Value, width);
02393           y2 = ParseLengthOrPercentage(lineObject.Attributes?["y2"]?.Value, height);
02394
02395           PresentationAttributes lineAttributes = InterpretPresentationAttributes(lineObject,
      attributes, width, height, diagonal, gpr, styleSheets, gradients);
02396
02397           bool hadClippingPath = ApplyClipPath(lineObject, gpr, width, height, diagonal, attributes,
      styleSheets, gradients);
02398
02399           string tag = lineObject.Attributes?["id"]?.Value;
02400
02401           if (lineAttributes.Stroke != null)
02402           {
02403               Brush strokeColour = lineAttributes.Stroke.MultiplyOpacity(lineAttributes.Opacity *
      lineAttributes.StrokeOpacity);
02404
02405               GraphicsPath path = new GraphicsPath().MoveTo(x1, y1).LineTo(x2, y2);
02406               GraphicsPath oldPath = path;
02407               double[,] transform = null;
02408
02409               if (lineAttributes.StrokeGradientNeedsTransform)
02410               {
02411                   (strokeColour, transform) = TransformBrush(strokeColour, new Rectangle(x1, y1, x2
      - x1, y2 - y1));
02412               }
02413
02414               if (transform != null)
02415               {
02416                   gpr.Save();
02417                   gpr.Transform(transform[0, 0], transform[1, 0], transform[0, 1], transform[1, 1],
      transform[0, 2], transform[1, 2]);
02418                   path = path.Transform(MatrixUtils.GetInverseTransformation(transform));
02419               }
02420
02421               gpr.StrokePath(path, strokeColour, lineAttributes.StrokeThickness,
      lineAttributes.LineCap, lineAttributes.LineJoin, lineAttributes.LineDash, tag:  tag);
02422
02423               if (transform != null)
02424               {
02425                   gpr.Restore();
02426                   path = oldPath;
02427               }
```

```
02428             }
02429
02430             if (hadClippingPath)
02431             {
02432                 gpr.Restore();
02433             }
02434
02435             if (lineAttributes.NeedsRestore)
02436             {
02437                 gpr.Restore();
02438             }
02439         }
02440
02441         private static void SetStyleAttributes(XmlNode obj, IEnumerable<Stylesheet> styleSheets)
02442         {
02443             string style = obj.Attributes?["style"]?.Value;
02444
02445             string classes = obj.Attributes?["class"]?.Value;
02446
02447             if (!string.IsNullOrEmpty(classes))
02448             {
02449                 string[] splitClasses = classes.Split(' ');
02450
02451                 foreach (string className in splitClasses)
02452                 {
02453                     if (!string.IsNullOrEmpty(className.Trim()))
02454                     {
02455                         foreach (Stylesheet sheet in styleSheets)
02456                         {
02457                             foreach (StyleRule rule in sheet.StyleRules)
02458                             {
02459                                 if (rule.SelectorText.Contains("." + className))
02460                                 {
02461                                     style = rule.Style.CssText + "; " + style;
02462                                 }
02463                             }
02464                         }
02465                     }
02466                 }
02467             }
02468
02469             if (!string.IsNullOrEmpty(style))
02470             {
02471                 string[] splitStyle = style.Split(';');
02472
02473                 for (int i = 0; i < splitStyle.Length; i++)
02474                 {
02475                     string[] styleCouple = splitStyle[i].Split(':');
02476
02477                     if (styleCouple.Length == 2)
02478                     {
02479                         string styleName = styleCouple[0].Trim();
02480                         string styleValue = styleCouple[1].Trim();
02481
02482                         ((XmlElement)obj).SetAttribute(styleName, styleValue);
02483                     }
02484                     else if (!string.IsNullOrWhiteSpace(splitStyle[i]))
02485                     {
02486                         throw new InvalidOperationException("The style specification is not valid:  "
      + splitStyle[i]);
02487                     }
02488                 }
02489             }
02490         }
02491
02492         internal static PresentationAttributes InterpretPresentationAttributes(XmlNode obj,
      PresentationAttributes parentPresentationAttributes, double width, double height, double diagonal,
      Graphics gpr, IEnumerable<Stylesheet> styleSheets, Dictionary<string, (bool, Brush)> gradients)
02493         {
02494             SetStyleAttributes(obj, styleSheets);
02495
02496             PresentationAttributes tbr = parentPresentationAttributes.Clone();
02497
02498             string stroke = obj.Attributes?["stroke"]?.Value;
02499             string strokeOpacity = obj.Attributes?["stroke-opacity"]?.Value;
02500             string fill = obj.Attributes?["fill"]?.Value;
02501             string fillOpacity = obj.Attributes?["fill-opacity"]?.Value;
02502             string currentColour = obj.Attributes?["colour"]?.Value;
02503             string strokeThickness = obj.Attributes?["stroke-width"]?.Value;
02504             string lineCap = obj.Attributes?["stroke-linecap"]?.Value;
02505             string lineJoin = obj.Attributes?["stroke-linejoin"]?.Value;
02506             string opacity = obj.Attributes?["opacity"]?.Value;
02507             string strokeDashArray = obj.Attributes?["stroke-dasharray"]?.Value;
02508             string strokeDashOffset = obj.Attributes?["stroke-dashoffset"]?.Value;
02509             string paintOrder = obj.Attributes?["paint-order"]?.Value;
02510
02511
```

```
02512             string xA = obj.Attributes?["x"]?.Value;
02513             string yA = obj.Attributes?["y"]?.Value;
02514             string wA = obj.Attributes?["width"]?.Value;
02515             string hA = obj.Attributes?["height"]?.Value;
02516 
02517             string transform = obj.Attributes?["transform"]?.Value;
02518 
02519             if (xA != null)
02520             {
02521                 tbr.X = ParseLengthOrPercentage(xA, width);
02522             }
02523 
02524             if (yA != null)
02525             {
02526                 tbr.Y = ParseLengthOrPercentage(yA, height);
02527             }
02528 
02529             if (wA != null)
02530             {
02531                 tbr.Width = ParseLengthOrPercentage(wA, width);
02532             }
02533 
02534             if (hA != null)
02535             {
02536                 tbr.Height = ParseLengthOrPercentage(hA, height);
02537             }
02538 
02539             if (stroke != null)
02540             {
02541                 if (stroke.Trim().StartsWith("url("))
02542                 {
02543                     string url = stroke.Trim().Substring(4);
02544                     if (url.EndsWith(")"))
02545                     {
02546                         url = url.Substring(0, url.Length - 1);
02547                     }
02548 
02549                     url = url.Trim();
02550 
02551                     if (url.StartsWith("#"))
02552                     {
02553                         url = url.Substring(1);
02554                         if (gradients.TryGetValue(url, out (bool, Brush) brush))
02555                         {
02556                             tbr.Stroke = brush.Item2;
02557                             tbr.StrokeGradientNeedsTransform = !brush.Item1;
02558                         }
02559                     }
02560                     else
02561                     {
02562                         tbr.Stroke = null;
02563                     }
02564                 }
02565                 else
02566                 {
02567                     tbr.Stroke = Colour.FromCSSString(stroke);
02568                 }
02569             }
02570 
02571             if (strokeOpacity != null)
02572             {
02573                 tbr.StrokeOpacity = ParseLengthOrPercentage(strokeOpacity, 1);
02574             }
02575 
02576             if (fill != null)
02577             {
02578                 if (fill.Trim().StartsWith("url("))
02579                 {
02580                     string url = fill.Trim().Substring(4);
02581                     if (url.EndsWith(")"))
02582                     {
02583                         url = url.Substring(0, url.Length - 1);
02584                     }
02585 
02586                     url = url.Trim();
02587 
02588                     if (url.StartsWith("#"))
02589                     {
02590                         url = url.Substring(1);
02591                         if (gradients.TryGetValue(url, out (bool, Brush) brush))
02592                         {
02593                             tbr.Fill = brush.Item2;
02594                             tbr.FillGradientNeedsTransform = !brush.Item1;
02595                         }
02596                     }
02597                     else
02598                     {
```

```
02599                           tbr.Fill = null;
02600                       }
02601                   }
02602                   else
02603                   {
02604                       tbr.Fill = Colour.FromCSSString(fill);
02605                   }
02606               }
02607
02608               if (fillOpacity != null)
02609               {
02610                   tbr.FillOpacity = ParseLengthOrPercentage(fillOpacity, 1);
02611               }
02612
02613               if (currentColour != null)
02614               {
02615                   tbr.CurrentColour = Colour.FromCSSString(currentColour);
02616               }
02617
02618               if (strokeThickness != null)
02619               {
02620                   tbr.StrokeThickness = ParseLengthOrPercentage(strokeThickness, diagonal);
02621               }
02622
02623               if (lineCap != null)
02624               {
02625                   if (lineCap.Equals("butt", StringComparison.OrdinalIgnoreCase))
02626                   {
02627                       tbr.LineCap = LineCaps.Butt;
02628                   }
02629                   else if (lineCap.Equals("round", StringComparison.OrdinalIgnoreCase))
02630                   {
02631                       tbr.LineCap = LineCaps.Round;
02632                   }
02633                   else if (lineCap.Equals("square", StringComparison.OrdinalIgnoreCase))
02634                   {
02635                       tbr.LineCap = LineCaps.Square;
02636                   }
02637               }
02638
02639               if (lineJoin != null)
02640               {
02641                   if (lineJoin.Equals("bevel", StringComparison.OrdinalIgnoreCase))
02642                   {
02643                       tbr.LineJoin = LineJoins.Bevel;
02644                   }
02645                   else if (lineJoin.Equals("miter", StringComparison.OrdinalIgnoreCase) ||
      lineJoin.Equals("miter-clip", StringComparison.OrdinalIgnoreCase))
02646                   {
02647                       tbr.LineJoin = LineJoins.Miter;
02648                   }
02649                   else if (lineJoin.Equals("round", StringComparison.OrdinalIgnoreCase))
02650                   {
02651                       tbr.LineJoin = LineJoins.Round;
02652                   }
02653               }
02654
02655               if (opacity != null)
02656               {
02657                   tbr.Opacity = ParseLengthOrPercentage(opacity, 1);
02658               }
02659
02660               if (strokeDashArray != null)
02661               {
02662                   if (strokeDashArray != "none")
02663                   {
02664                       double[] parsedArray = ParseListOfDoubles(strokeDashArray, diagonal);
02665
02666                       tbr.LineDash = new LineDash(parsedArray, tbr.LineDash.Phase);
02667                   }
02668                   else
02669                   {
02670                       tbr.LineDash = LineDash.SolidLine;
02671                   }
02672               }
02673
02674               if (strokeDashOffset != null)
02675               {
02676                   tbr.LineDash = new LineDash(tbr.LineDash.DashArray,
      ParseLengthOrPercentage(strokeDashOffset, diagonal));
02677               }
02678
02679               if (paintOrder != null)
02680               {
02681                   if (paintOrder.Equals("normal", StringComparison.OrdinalIgnoreCase))
02682                   {
02683                       tbr.StrokeFirst = false;
```

```
02684                 }
02685                 else
02686                 {
02687                     if (paintOrder.IndexOf("stroke", StringComparison.OrdinalIgnoreCase) >= 0 &&
      (paintOrder.IndexOf("fill", StringComparison.OrdinalIgnoreCase) < 0 || paintOrder.IndexOf("fill",
      StringComparison.OrdinalIgnoreCase) > paintOrder.IndexOf("stroke",
      StringComparison.OrdinalIgnoreCase)))
02688                     {
02689                         tbr.StrokeFirst = true;
02690                     }
02691                     else
02692                     {
02693                         tbr.StrokeFirst = false;
02694                     }
02695                 }
02696             }
02697 
02698             if (transform != null)
02699             {
02700                 gpr.Save();
02701                 tbr.NeedsRestore = true;
02702 
02703                 string[] transforms = ParseListOfTransforms(transform);
02704 
02705                 for (int i = 0; i < transforms.Length; i++)
02706                 {
02707                     if (transforms[i].Equals("matrix", StringComparison.OrdinalIgnoreCase))
02708                     {
02709                         double a = ParseLengthOrPercentage(transforms[i + 1], 1);
02710                         double b = ParseLengthOrPercentage(transforms[i + 2], 1);
02711                         double c = ParseLengthOrPercentage(transforms[i + 3], 1);
02712                         double d = ParseLengthOrPercentage(transforms[i + 4], 1);
02713                         double e = ParseLengthOrPercentage(transforms[i + 5], 1);
02714                         double f = ParseLengthOrPercentage(transforms[i + 6], 1);
02715 
02716                         gpr.Transform(a, b, c, d, e, f);
02717                         i += 6;
02718                     }
02719                     else if (transforms[i].Equals("translate", StringComparison.OrdinalIgnoreCase))
02720                     {
02721                         double x = ParseLengthOrPercentage(transforms[i + 1], 1);
02722 
02723                         double y;
02724 
02725                         if (i < transforms.Length - 2 && !double.IsNaN(y =
      ParseLengthOrPercentage(transforms[i + 2], 1)))
02726                         {
02727                             gpr.Translate(x, y);
02728                             i += 2;
02729                         }
02730                         else
02731                         {
02732                             gpr.Translate(x, 0);
02733                             i++;
02734                         }
02735                     }
02736                     else if (transforms[i].Equals("scale", StringComparison.OrdinalIgnoreCase))
02737                     {
02738                         double x = ParseLengthOrPercentage(transforms[i + 1], 1);
02739 
02740                         double y;
02741 
02742                         if (i < transforms.Length - 2 && !double.IsNaN(y =
      ParseLengthOrPercentage(transforms[i + 2], 1)))
02743                         {
02744                             gpr.Scale(x, y);
02745                             i += 2;
02746                         }
02747                         else
02748                         {
02749                             gpr.Scale(x, x);
02750                             i++;
02751                         }
02752                     }
02753                     else if (transforms[i].Equals("rotate", StringComparison.OrdinalIgnoreCase))
02754                     {
02755                         double a = ParseLengthOrPercentage(transforms[i + 1], 1) * Math.PI / 180;
02756 
02757                         double x, y;
02758 
02759                         if (i < transforms.Length - 3 && !double.IsNaN(x =
      ParseLengthOrPercentage(transforms[i + 2], 1)) && !double.IsNaN(y =
      ParseLengthOrPercentage(transforms[i + 3], 1)))
02760                         {
02761                             gpr.RotateAt(a, new Point(x, y));
02762                             i += 2;
02763                         }
```

```
02764                           else
02765                           {
02766                               gpr.Rotate(a);
02767                               i++;
02768                           }
02769                       }
02770                       else if (transforms[i].Equals("skewX", StringComparison.OrdinalIgnoreCase))
02771                       {
02772                           double psi = ParseLengthOrPercentage(transforms[i + 1], 1) * Math.PI / 180;
02773
02774                           gpr.Transform(1, 0, Math.Tan(psi), 1, 0, 0);
02775
02776                           i++;
02777                       }
02778                       else if (transforms[i].Equals("skewY", StringComparison.OrdinalIgnoreCase))
02779                       {
02780                           double psi = ParseLengthOrPercentage(transforms[i + 1], 1) * Math.PI / 180;
02781
02782                           gpr.Transform(1, Math.Tan(psi), 0, 1, 0, 0);
02783
02784                           i++;
02785                       }
02786                   }
02787               }
02788
02789               return tbr;
02790           }
02791
02792           private static double ParseLengthOrPercentage(string value, double total, double defaultValue
      = 0)
02793           {
02794               if (value != null)
02795               {
02796                   if (value.Contains("%"))
02797                   {
02798                       value = value.Replace("%", "");
02799                       return double.Parse(value, System.Globalization.CultureInfo.InvariantCulture) *
      total / 100;
02800                   }
02801                   else if (double.TryParse(value.Replace("px", "").Replace("pt", ""),
      System.Globalization.NumberStyles.Any, System.Globalization.CultureInfo.InvariantCulture, out double
      result))
02802                   {
02803                       return result;
02804                   }
02805                   else
02806                   {
02807                       string cleanedNumber = Regexes.NumberRegex.Match(value).Value;
02808                       return double.Parse(cleanedNumber,
      System.Globalization.CultureInfo.InvariantCulture);
02809                   }
02810               }
02811               else
02812               {
02813                   return defaultValue;
02814               }
02815           }
02816
02817           internal class PresentationAttributes
02818           {
02819               public Dictionary<string, FontFamily> EmbeddedFonts;
02820
02821               public Brush Stroke = null;
02822               public double StrokeOpacity = 1;
02823               public Brush Fill = Colour.FromRgb(0, 0, 0);
02824               public double FillOpacity = 1;
02825               public Brush CurrentColour = null;
02826               public double StrokeThickness = 1;
02827               public LineCaps LineCap = LineCaps.Butt;
02828               public LineJoins LineJoin = LineJoins.Miter;
02829               public double Opacity = 1;
02830               public LineDash LineDash = new LineDash(0, 0, 0);
02831               public bool NeedsRestore = false;
02832               public bool StrokeFirst = false;
02833               public double X;
02834               public double Y;
02835               public double Width;
02836               public double Height;
02837               public bool StrokeGradientNeedsTransform = false;
02838               public bool FillGradientNeedsTransform = false;
02839
02840               public PresentationAttributes Clone()
02841               {
02842                   return new PresentationAttributes()
02843                   {
02844                       EmbeddedFonts = this.EmbeddedFonts,
02845
```

```
02846                     Stroke = this.Stroke,
02847                     StrokeOpacity = this.StrokeOpacity,
02848                     Fill = this.Fill,
02849                     FillOpacity = this.FillOpacity,
02850                     CurrentColour = this.CurrentColour,
02851                     StrokeThickness = this.StrokeThickness,
02852                     LineCap = this.LineCap,
02853                     LineJoin = this.LineJoin,
02854                     Opacity = this.Opacity,
02855                     LineDash = this.LineDash,
02856                     StrokeFirst = this.StrokeFirst,
02857                     X = this.X,
02858                     Y = this.Y,
02859                     Width = this.Width,
02860                     Height = this.Height
02861                 };
02862             }
02863         }
02864
02865         private static class Regexes
02866         {
02867             public static Regex ListSeparator = new Regex("[ \\t\\n\\r\\f]*,[ \\t\\n\\r\\f]*|[
      \\t\\n\\r\\f]+", RegexOptions.Compiled);
02868             public static Regex FontFamilySeparator = new Regex("(?:^|,)(\"(?:[^\"])*\"|[^,]*)",
      RegexOptions.Compiled);
02869             public static Regex NumberRegex = new Regex(@"^[-+]?[0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?",
      RegexOptions.Compiled);
02870         }
02871
02872         private static double[] ParseListOfDoubles(string value, double?  total = null)
02873         {
02874             if (value == null)
02875             {
02876                 return null;
02877             }
02878
02879             string[] splitValue = Regexes.ListSeparator.Split(value);
02880             double[] tbr = new double[splitValue.Length];
02881
02882             for (int i = 0; i < splitValue.Length; i++)
02883             {
02884                 if (total == null)
02885                 {
02886                     tbr[i] = double.Parse(splitValue[i],
      System.Globalization.CultureInfo.InvariantCulture);
02887                 }
02888                 else
02889                 {
02890                     tbr[i] = ParseLengthOrPercentage(splitValue[i], total.Value);
02891                 }
02892             }
02893
02894             return tbr;
02895         }
02896
02897         private static string[] ParseListOfTransforms(string value)
02898         {
02899             if (value == null)
02900             {
02901                 return null;
02902             }
02903
02904             string[] splitValue = Regexes.ListSeparator.Split(value.Replace("(", " ").Replace(")", "
      ").Trim());
02905
02906             return splitValue;
02907         }
02908
02909         private static List<KeyValuePair<string, FontFamily» GetEmbeddedFonts(string styleBlock)
02910         {
02911             StringReader sr = new StringReader(styleBlock);
02912
02913             List<KeyValuePair<string, FontFamily» tbr = new List<KeyValuePair<string, FontFamily»();
02914
02915             while (sr.Peek() >= 0)
02916             {
02917                 string token = ReadCSSToken(sr);
02918
02919                 if (token.Equals("@font-face", StringComparison.OrdinalIgnoreCase))
02920                 {
02921                     List<string> tokens = new List<string>();
02922
02923                     while (!token.Equals("}", StringComparison.OrdinalIgnoreCase))
02924                     {
02925                         token = ReadCSSToken(sr);
02926                         tokens.Add(token);
02927                     }
```

```
02928
02929                     KeyValuePair<string, FontFamily>?  fontFace = ParseFontFaceBlock(tokens);
02930
02931                     if (fontFace != null)
02932                     {
02933                         tbr.Add(fontFace.Value);
02934                     }
02935                 }
02936             }
02937
02938             return tbr;
02939         }
02940
02941         private static IEnumerable<KeyValuePair<string, IFilter» GetFilters(XmlNode definitionsNode,
      List<Stylesheet> styleSheets)
02942         {
02943             Dictionary<string, IFilter> tbr = new Dictionary<string, IFilter>();
02944
02945             foreach (XmlNode definition in definitionsNode.ChildNodes)
02946             {
02947                 if (definition.Name.Equals("filter", StringComparison.OrdinalIgnoreCase))
02948                 {
02949                     XmlElement filter = (XmlElement)definition;
02950
02951                     string id = filter.GetAttribute("id");
02952
02953                     List<ILocationInvariantFilter> filterElements = new
      List<ILocationInvariantFilter>();
02954
02955                     foreach (XmlNode filterDefinition in definition.ChildNodes)
02956                     {
02957                         if (filterDefinition.Name.Equals("feGaussianBlur",
      StringComparison.OrdinalIgnoreCase))
02958                         {
02959                             XmlElement actualFilter = (XmlElement)filterDefinition;
02960
02961                             string stdDeviation = actualFilter.GetAttribute("stdDeviation");
02962
02963                             filterElements.Add(new GaussianBlurFilter(double.Parse(stdDeviation,
      System.Globalization.CultureInfo.InvariantCulture)));
02964                         }
02965                         else if (filterDefinition.Name.Equals("feColorMatrix",
      StringComparison.OrdinalIgnoreCase))
02966                         {
02967                             XmlElement actualFilter = (XmlElement)filterDefinition;
02968
02969                             string type = actualFilter.GetAttribute("type");
02970
02971                             if (type.Equals("matrix", StringComparison.OrdinalIgnoreCase))
02972                             {
02973                                 string values = actualFilter.GetAttribute("values");
02974                                 double[] parsedValues = (from el in
      System.Text.RegularExpressions.Regex.Split(values, "\\s") select double.Parse(el.Trim(),
      System.Globalization.CultureInfo.InvariantCulture)).ToArray();
02975
02976                                 if (parsedValues.Length == 20)
02977                                 {
02978                                     double[,] matrix = new double[5, 5];
02979                                     matrix[4, 4] = 1;
02980
02981                                     for (int i = 0; i < 20; i++)
02982                                     {
02983                                         int y = i / 5;
02984                                         int x = i % 5;
02985
02986                                         matrix[y, x] = parsedValues[i];
02987                                     }
02988
02989                                     filterElements.Add(new ColourMatrixFilter(new
      ColourMatrix(matrix)));
02990                                 }
02991                             }
02992                         }
02993                     }
02994
02995                     if (filterElements.Count > 0)
02996                     {
02997                         if (filterElements.Count == 1)
02998                         {
02999                             tbr.Add(id, filterElements[0]);
03000                         }
03001                         else
03002                         {
03003                             tbr.Add(id, new CompositeLocationInvariantFilter(filterElements));
03004                         }
03005                     }
03006                 }
```

```
03007             }
03008
03009             return tbr;
03010         }
03011
03012         private static IEnumerable<KeyValuePair<string, XmlNode» GetMasks(XmlNode definitionsNode,
      List<Stylesheet> styleSheets)
03013         {
03014             Dictionary<string, XmlNode> tbr = new Dictionary<string, XmlNode>();
03015
03016             foreach (XmlNode definition in definitionsNode.ChildNodes)
03017             {
03018                 if (definition.Name.Equals("mask", StringComparison.OrdinalIgnoreCase))
03019                 {
03020                     XmlElement mask = (XmlElement)definition;
03021
03022                     string id = mask.GetAttribute("id");
03023
03024                     tbr.Add(id, definition);
03025                 }
03026             }
03027
03028             return tbr;
03029         }
03030
03031         private static IEnumerable<KeyValuePair<string, (bool, Brush)» GetGradients(XmlNode
      definitionsNode, List<Stylesheet> styleSheets)
03032         {
03033             Dictionary<string, (bool, Brush)> tbr = new Dictionary<string, (bool, Brush)>();
03034             Dictionary<string, XmlNodeList> stopLists = new Dictionary<string, XmlNodeList>();
03035
03036             foreach (XmlNode definition in definitionsNode.ChildNodes)
03037             {
03038                 if (definition.Name.Equals("linearGradient", StringComparison.OrdinalIgnoreCase))
03039                 {
03040                     XmlElement gradient = (XmlElement)definition;
03041
03042                     string id = gradient.GetAttribute("id");
03043
03044                     bool userSpaceOnUse = gradient.GetAttribute("gradientUnits") == "userSpaceOnUse";
03045
03046                     double x1;
03047                     double y1;
03048                     double x2;
03049                     double y2;
03050
03051                     if (!(gradient.HasAttribute("x1") && double.TryParse(gradient.GetAttribute("x1"),
      System.Globalization.NumberStyles.Any, System.Globalization.CultureInfo.InvariantCulture, out x1)))
03052                     {
03053                         x1 = 0;
03054                     }
03055
03056                     if (!(gradient.HasAttribute("y1") && double.TryParse(gradient.GetAttribute("y1"),
      System.Globalization.NumberStyles.Any, System.Globalization.CultureInfo.InvariantCulture, out y1)))
03057                     {
03058                         y1 = 0;
03059                     }
03060
03061                     if (!(gradient.HasAttribute("x2") && double.TryParse(gradient.GetAttribute("x2"),
      System.Globalization.NumberStyles.Any, System.Globalization.CultureInfo.InvariantCulture, out x2)))
03062                     {
03063                         x2 = 0;
03064                     }
03065
03066                     if (!(gradient.HasAttribute("y2") && double.TryParse(gradient.GetAttribute("y2"),
      System.Globalization.NumberStyles.Any, System.Globalization.CultureInfo.InvariantCulture, out y2)))
03067                     {
03068                         y2 = 0;
03069                     }
03070
03071                     List<GradientStop> gradientStops = new List<GradientStop>();
03072
03073                     XmlNodeList childNodes;
03074
03075                     if (gradient.HasAttribute("xlink:href"))
03076                     {
03077                         string refId = gradient.GetAttribute("xlink:href").Trim();
03078
03079                         if (refId.StartsWith("#"))
03080                         {
03081                             refId = refId.Substring(1);
03082
03083                             if (!stopLists.TryGetValue(refId, out childNodes))
03084                             {
03085                                 childNodes = gradient.ChildNodes;
03086                             }
03087                         }
```

```
03088                           else
03089                           {
03090                               childNodes = gradient.ChildNodes;
03091                           }
03092                       }
03093                       else
03094                       {
03095                           childNodes = gradient.ChildNodes;
03096                       }
03097
03098                       stopLists[id] = childNodes;
03099
03100                       foreach (XmlNode stopNode in childNodes)
03101                       {
03102                           if (stopNode.Name.Equals("stop", StringComparison.OrdinalIgnoreCase))
03103                           {
03104                               SetStyleAttributes(stopNode, styleSheets);
03105
03106                               XmlElement stop = (XmlElement)stopNode;
03107
03108                               double offset = 0;
03109                               double opacity = 1;
03110
03111                               if (stop.HasAttribute("offset"))
03112                               {
03113                                   offset = ParseLengthOrPercentage(stop.GetAttribute("offset"), 1);
03114                               }
03115
03116                               if (stop.HasAttribute("stop-opacity"))
03117                               {
03118                                   opacity = ParseLengthOrPercentage(stop.GetAttribute("stop-opacity"),
     1);
03119                               }
03120
03121                               Colour stopColour = Colour.FromRgba(0, 0, 0, 0);
03122
03123                               if (stop.HasAttribute("stop-color"))
03124                               {
03125                                   stopColour = (Colour.FromCSSString(stop.GetAttribute("stop-color")) ??
     stopColour).WithAlpha(opacity);
03126                               }
03127
03128                               gradientStops.Add(new GradientStop(stopColour, offset));
03129                           }
03130                       }
03131
03132                       if (gradient.HasAttribute("gradientTransform"))
03133                       {
03134                           string transform = gradient.GetAttribute("gradientTransform");
03135                           string[] transforms = ParseListOfTransforms(transform);
03136
03137                           double[,] transformMatrix = MatrixUtils.Identity;
03138
03139                           for (int i = 0; i < transforms.Length; i++)
03140                           {
03141                               if (transforms[i].Equals("matrix", StringComparison.OrdinalIgnoreCase))
03142                               {
03143                                   double a = ParseLengthOrPercentage(transforms[i + 1], 1);
03144                                   double b = ParseLengthOrPercentage(transforms[i + 2], 1);
03145                                   double c = ParseLengthOrPercentage(transforms[i + 3], 1);
03146                                   double d = ParseLengthOrPercentage(transforms[i + 4], 1);
03147                                   double e = ParseLengthOrPercentage(transforms[i + 5], 1);
03148                                   double f = ParseLengthOrPercentage(transforms[i + 6], 1);
03149
03150                                   transformMatrix = MatrixUtils.Multiply(transformMatrix, new double[,]
     { { a, c, e }, { b, d, f }, { 0, 0, 1 } });
03151
03152                                   i += 6;
03153                               }
03154                               else if (transforms[i].Equals("translate",
     StringComparison.OrdinalIgnoreCase))
03155                               {
03156                                   double x = ParseLengthOrPercentage(transforms[i + 1], 1);
03157
03158                                   double y;
03159
03160                                   if (i < transforms.Length - 2 && !double.IsNaN(y =
     ParseLengthOrPercentage(transforms[i + 2], 1)))
03161                                   {
03162                                       transformMatrix = MatrixUtils.Translate(transformMatrix, x, y);
03163                                       i += 2;
03164                                   }
03165                                   else
03166                                   {
03167                                       transformMatrix = MatrixUtils.Translate(transformMatrix, x, 0);
03168                                       i++;
03169                                   }
```

```
03170                           }
03171                           else if (transforms[i].Equals("scale",
      StringComparison.OrdinalIgnoreCase))
03172                           {
03173                               double x = ParseLengthOrPercentage(transforms[i + 1], 1);
03174
03175                               double y;
03176
03177                               if (i < transforms.Length - 2 && !double.IsNaN(y =
      ParseLengthOrPercentage(transforms[i + 2], 1)))
03178                               {
03179                                   transformMatrix = MatrixUtils.Scale(transformMatrix, x, y);
03180                                   i += 2;
03181                               }
03182                               else
03183                               {
03184                                   transformMatrix = MatrixUtils.Scale(transformMatrix, x, x);
03185                                   i++;
03186                               }
03187                           }
03188                           else if (transforms[i].Equals("rotate",
      StringComparison.OrdinalIgnoreCase))
03189                           {
03190                               double a = ParseLengthOrPercentage(transforms[i + 1], 1) * Math.PI /
      180;
03191
03192                               double x, y;
03193
03194                               if (i < transforms.Length - 3 && !double.IsNaN(x =
      ParseLengthOrPercentage(transforms[i + 2], 1)) && !double.IsNaN(y =
      ParseLengthOrPercentage(transforms[i + 3], 1)))
03195                               {
03196                                   transformMatrix = MatrixUtils.Translate(transformMatrix, x, y);
03197                                   transformMatrix = MatrixUtils.Rotate(transformMatrix, a);
03198                                   transformMatrix = MatrixUtils.Translate(transformMatrix, -x, -y);
03199                                   i += 2;
03200                               }
03201                               else
03202                               {
03203                                   transformMatrix = MatrixUtils.Rotate(transformMatrix, a);
03204                                   i++;
03205                               }
03206                           }
03207                           else if (transforms[i].Equals("skewX",
      StringComparison.OrdinalIgnoreCase))
03208                           {
03209                               double psi = ParseLengthOrPercentage(transforms[i + 1], 1) * Math.PI /
      180;
03210
03211                               transformMatrix = MatrixUtils.Multiply(transformMatrix, new double[,]
      { { 1, Math.Tan(psi), 0 }, { 0, 1, 0 }, { 0, 0, 1 } });
03212                               i++;
03213                           }
03214                           else if (transforms[i].Equals("skewY",
      StringComparison.OrdinalIgnoreCase))
03215                           {
03216                               double psi = ParseLengthOrPercentage(transforms[i + 1], 1) * Math.PI /
      180;
03217
03218                               transformMatrix = MatrixUtils.Multiply(transformMatrix, new double[,]
      { { 1, 0, 0 }, { Math.Tan(psi), 1, 0 }, { 0, 0, 1 } });
03219                               i++;
03220                           }
03221                       }
03222
03223                       double[] start = MatrixUtils.Multiply(transformMatrix, new double[] { x1, y1
      });
03224                       double[] end = MatrixUtils.Multiply(transformMatrix, new double[] { x2, y2 });
03225
03226                       x1 = start[0];
03227                       y1 = start[1];
03228                       x2 = end[0];
03229                       y2 = end[1];
03230                   }
03231
03232                   tbr.Add(id, (userSpaceOnUse, new LinearGradientBrush(new Point(x1, y1), new
      Point(x2, y2), gradientStops)));
03233               }
03234               else if (definition.Name.Equals("radialGradient", StringComparison.OrdinalIgnoreCase))
03235               {
03236                   XmlElement gradient = (XmlElement)definition;
03237
03238                   string id = gradient.GetAttribute("id");
03239
03240                   bool userSpaceOnUse = gradient.GetAttribute("gradientUnits") == "userSpaceOnUse";
03241
03242                   double cx;
```

```
03243                     double cy;
03244                     double r;
03245 
03246                     if (!(gradient.HasAttribute("cx") && double.TryParse(gradient.GetAttribute("cx"),
      System.Globalization.NumberStyles.Any, System.Globalization.CultureInfo.InvariantCulture, out cx)))
03247                     {
03248                         cx = 0;
03249                     }
03250 
03251                     if (!(gradient.HasAttribute("cy") && double.TryParse(gradient.GetAttribute("cy"),
      System.Globalization.NumberStyles.Any, System.Globalization.CultureInfo.InvariantCulture, out cy)))
03252                     {
03253                         cy = 0;
03254                     }
03255 
03256                     if (!(gradient.HasAttribute("r") && double.TryParse(gradient.GetAttribute("r"),
      System.Globalization.NumberStyles.Any, System.Globalization.CultureInfo.InvariantCulture, out r)))
03257                     {
03258                         r = 0;
03259                     }
03260 
03261                     double fx;
03262                     double fy;
03263 
03264                     if (!(gradient.HasAttribute("fx") && double.TryParse(gradient.GetAttribute("fx"),
      System.Globalization.NumberStyles.Any, System.Globalization.CultureInfo.InvariantCulture, out fx)))
03265                     {
03266                         fx = cx;
03267                     }
03268 
03269                     if (!(gradient.HasAttribute("fy") && double.TryParse(gradient.GetAttribute("fy"),
      System.Globalization.NumberStyles.Any, System.Globalization.CultureInfo.InvariantCulture, out fy)))
03270                     {
03271                         fy = cy;
03272                     }
03273 
03274                     List<GradientStop> gradientStops = new List<GradientStop>();
03275 
03276                     XmlNodeList childNodes;
03277 
03278                     if (gradient.HasAttribute("xlink:href"))
03279                     {
03280                         string refId = gradient.GetAttribute("xlink:href").Trim();
03281 
03282                         if (refId.StartsWith("#"))
03283                         {
03284                             refId = refId.Substring(1);
03285 
03286                             if (!stopLists.TryGetValue(refId, out childNodes))
03287                             {
03288                                 childNodes = gradient.ChildNodes;
03289                             }
03290                         }
03291                         else
03292                         {
03293                             childNodes = gradient.ChildNodes;
03294                         }
03295                     }
03296                     else
03297                     {
03298                         childNodes = gradient.ChildNodes;
03299                     }
03300 
03301                     stopLists[id] = childNodes;
03302 
03303                     foreach (XmlNode stopNode in childNodes)
03304                     {
03305                         if (stopNode.Name.Equals("stop", StringComparison.OrdinalIgnoreCase))
03306                         {
03307                             SetStyleAttributes(stopNode, styleSheets);
03308 
03309                             XmlElement stop = (XmlElement)stopNode;
03310 
03311                             double offset = 0;
03312                             double opacity = 1;
03313 
03314                             if (stop.HasAttribute("offset"))
03315                             {
03316                                 offset = ParseLengthOrPercentage(stop.GetAttribute("offset"), 1);
03317                             }
03318 
03319                             if (stop.HasAttribute("stop-opacity"))
03320                             {
03321                                 opacity = ParseLengthOrPercentage(stop.GetAttribute("stop-opacity"),
      1);
03322                             }
03323 
```

```
03324                               Colour stopColour = Colour.FromRgba(0, 0, 0, 0);
03325
03326                               if (stop.HasAttribute("stop-color"))
03327                               {
03328                                   stopColour = (Colour.FromCSSString(stop.GetAttribute("stop-color")) ??
      stopColour).WithAlpha(opacity);
03329                               }
03330
03331                               gradientStops.Add(new GradientStop(stopColour, offset));
03332                           }
03333                       }
03334
03335                       if (gradient.HasAttribute("gradientTransform"))
03336                       {
03337                           string transform = gradient.GetAttribute("gradientTransform");
03338                           string[] transforms = ParseListOfTransforms(transform);
03339
03340                           double[,] transformMatrix = MatrixUtils.Identity;
03341
03342                           for (int i = 0; i < transforms.Length; i++)
03343                           {
03344                               if (transforms[i].Equals("matrix", StringComparison.OrdinalIgnoreCase))
03345                               {
03346                                   double a = ParseLengthOrPercentage(transforms[i + 1], 1);
03347                                   double b = ParseLengthOrPercentage(transforms[i + 2], 1);
03348                                   double c = ParseLengthOrPercentage(transforms[i + 3], 1);
03349                                   double d = ParseLengthOrPercentage(transforms[i + 4], 1);
03350                                   double e = ParseLengthOrPercentage(transforms[i + 5], 1);
03351                                   double f = ParseLengthOrPercentage(transforms[i + 6], 1);
03352
03353                                   transformMatrix = MatrixUtils.Multiply(transformMatrix, new double[,]
      { { a, c, e }, { b, d, f }, { 0, 0, 1 } });
03354
03355                                   i += 6;
03356                               }
03357                               else if (transforms[i].Equals("translate",
      StringComparison.OrdinalIgnoreCase))
03358                               {
03359                                   double x = ParseLengthOrPercentage(transforms[i + 1], 1);
03360
03361                                   double y;
03362
03363                                   if (i < transforms.Length - 2 && !double.IsNaN(y =
      ParseLengthOrPercentage(transforms[i + 2], 1)))
03364                                   {
03365                                       transformMatrix = MatrixUtils.Translate(transformMatrix, x, y);
03366                                       i += 2;
03367                                   }
03368                                   else
03369                                   {
03370                                       transformMatrix = MatrixUtils.Translate(transformMatrix, x, 0);
03371                                       i++;
03372                                   }
03373                               }
03374                               else if (transforms[i].Equals("scale",
      StringComparison.OrdinalIgnoreCase))
03375                               {
03376                                   double x = ParseLengthOrPercentage(transforms[i + 1], 1);
03377
03378                                   double y;
03379
03380                                   if (i < transforms.Length - 2 && !double.IsNaN(y =
      ParseLengthOrPercentage(transforms[i + 2], 1)))
03381                                   {
03382                                       transformMatrix = MatrixUtils.Scale(transformMatrix, x, y);
03383                                       i += 2;
03384                                   }
03385                                   else
03386                                   {
03387                                       transformMatrix = MatrixUtils.Scale(transformMatrix, x, x);
03388                                       i++;
03389                                   }
03390                               }
03391                               else if (transforms[i].Equals("rotate",
      StringComparison.OrdinalIgnoreCase))
03392                               {
03393                                   double a = ParseLengthOrPercentage(transforms[i + 1], 1) * Math.PI /
      180;
03394
03395                                   double x, y;
03396
03397                                   if (i < transforms.Length - 3 && !double.IsNaN(x =
      ParseLengthOrPercentage(transforms[i + 2], 1)) && !double.IsNaN(y =
      ParseLengthOrPercentage(transforms[i + 3], 1)))
03398                                   {
03399                                       transformMatrix = MatrixUtils.Translate(transformMatrix, x, y);
03400                                       transformMatrix = MatrixUtils.Rotate(transformMatrix, a);
```

```
03401                                   transformMatrix = MatrixUtils.Translate(transformMatrix, -x, -y);
03402                                   i += 2;
03403                               }
03404                               else
03405                               {
03406                                   transformMatrix = MatrixUtils.Rotate(transformMatrix, a);
03407                                   i++;
03408                               }
03409                           }
03410                           else if (transforms[i].Equals("skewX",
      StringComparison.OrdinalIgnoreCase))
03411                           {
03412                               double psi = ParseLengthOrPercentage(transforms[i + 1], 1) * Math.PI /
      180;
03413
03414                               transformMatrix = MatrixUtils.Multiply(transformMatrix, new double[,]
      { { 1, Math.Tan(psi), 0 }, { 0, 1, 0 }, { 0, 0, 1 } });
03415                               i++;
03416                           }
03417                           else if (transforms[i].Equals("skewY",
      StringComparison.OrdinalIgnoreCase))
03418                           {
03419                               double psi = ParseLengthOrPercentage(transforms[i + 1], 1) * Math.PI /
      180;
03420
03421                               transformMatrix = MatrixUtils.Multiply(transformMatrix, new double[,]
      { { 1, 0, 0 }, { Math.Tan(psi), 1, 0 }, { 0, 0, 1 } });
03422                               i++;
03423                           }
03424                       }
03425
03426                       double determinant = transformMatrix[0, 0] * (transformMatrix[1, 1] *
      transformMatrix[2, 2] - transformMatrix[1, 2] * transformMatrix[2, 1]) -
03427                           transformMatrix[0, 1] * (transformMatrix[1, 0] * transformMatrix[2, 2] -
      transformMatrix[1, 2] * transformMatrix[2, 0]) +
03428                           transformMatrix[0, 2] * (transformMatrix[1, 0] * transformMatrix[2, 1] -
      transformMatrix[1, 1] * transformMatrix[2, 0]);
03429
03430                       double[] focus = MatrixUtils.Multiply(transformMatrix, new double[] { fx, fy
      });
03431                       double[] centre = MatrixUtils.Multiply(transformMatrix, new double[] { cx, cy
      });
03432
03433                       fx = focus[0];
03434                       fy = focus[1];
03435                       cx = centre[0];
03436                       cy = centre[1];
03437                       r = r * Math.Sqrt(determinant);
03438                   }
03439
03440                   tbr.Add(id, (userSpaceOnUse, new RadialGradientBrush(new Point(fx, fy), new
      Point(cx, cy), r, gradientStops)));
03441               }
03442           }
03443
03444
03445           return tbr;
03446       }
03447
03448       private static KeyValuePair<string, FontFamily>?  ParseFontFaceBlock(List<string> tokens)
03449       {
03450           int fontFamilyInd = tokens.IndexOf("font-family");
03451           string fontFamilyName = tokens[fontFamilyInd + 2].Trim().Trim('"').Trim();
03452
03453           int srcInd = tokens.IndexOf("src");
03454           string src = tokens[srcInd + 2];
03455
03456           string mimeType = src.Substring(src.IndexOf("data:") + 5);
03457           mimeType = mimeType.Substring(0, mimeType.IndexOf(";"));
03458
03459           if (mimeType.Equals("font/ttf", StringComparison.OrdinalIgnoreCase) ||
      mimeType.Equals("font/truetype", StringComparison.OrdinalIgnoreCase) ||
      mimeType.Equals("application/x-font-ttf", StringComparison.OrdinalIgnoreCase))
03460           {
03461               src = src.Substring(src.IndexOf("base64,") + 7);
03462               src = src.TrimEnd(')').TrimEnd('\"').TrimEnd(')');
03463               byte[] fontBytes = Convert.FromBase64String(src);
03464
03465               string tempFile = Path.GetTempFileName();
03466
03467               File.WriteAllBytes(tempFile, fontBytes);
03468
03469               FontFamily family = FontFamily.ResolveFontFamily(tempFile);
03470               return new KeyValuePair<string, FontFamily>(fontFamilyName, family);
03471           }
03472
03473           return null;
```

```
03474         }
03475
03476         private const string CSSDelimiters = ":;,{}";
03477
03478         private static string ReadCSSToken(StringReader reader)
03479         {
03480             StringBuilder tbr = new StringBuilder();
03481
03482             bool openQuotes = false;
03483             int openParentheses = 0;
03484
03485             int c = reader.Read();
03486             if (c >= 0)
03487             {
03488                 tbr.Append((char)c);
03489
03490                 if ((char)c == '"')
03491                 {
03492                     openQuotes = !openQuotes;
03493                 }
03494
03495                 if ((char)c == '(')
03496                 {
03497                     openParentheses++;
03498                 }
03499                 if ((char)c == ')')
03500                 {
03501                     openParentheses--;
03502                 }
03503
03504
03505                 while (c >= 0 && (!CSSDelimiters.Contains((char)c) || openQuotes || openParentheses >
      0))
03506                 {
03507                     c = reader.Read();
03508                     tbr.Append((char)c);
03509                     if ((char)c == '"')
03510                     {
03511                         openQuotes = !openQuotes;
03512                     }
03513                     if ((char)c == '(')
03514                     {
03515                         openParentheses++;
03516                     }
03517                     if ((char)c == ')')
03518                     {
03519                         openParentheses--;
03520                     }
03521                     c = reader.Peek();
03522                 }
03523             }
03524
03525             string val = tbr.ToString().Trim();
03526
03527             return (string.IsNullOrEmpty(val) && c >= 0) ?  ReadCSSToken(reader) :  val;
03528         }
03529     }
03530
03531     internal class PathTransformerGraphicsContext :  IGraphicsContext
03532     {
03533         public GraphicsPath CurrentPath = new GraphicsPath();
03534         public double[,] TransformMatrix = MatrixUtils.Identity;
03535
03536         private Stack<double[,]> transformMatrices;
03537         public PathTransformerGraphicsContext()
03538         {
03539             transformMatrices = new Stack<double[,]>();
03540             transformMatrices.Push((double[,])TransformMatrix.Clone());
03541         }
03542
03543         public void CubicBezierTo(double p1X, double p1Y, double p2X, double p2Y, double p3X, double
      p3Y)
03544         {
03545             double[] p1 = MatrixUtils.Multiply(TransformMatrix, new double[] { p1X, p1Y });
03546             double[] p2 = MatrixUtils.Multiply(TransformMatrix, new double[] { p2X, p2Y });
03547             double[] p3 = MatrixUtils.Multiply(TransformMatrix, new double[] { p3X, p3Y });
03548
03549             CurrentPath.CubicBezierTo(p1[0], p1[1], p2[0], p2[1], p3[0], p3[1]);
03550         }
03551
03552         public void LineTo(double x, double y)
03553         {
03554             double[] p = MatrixUtils.Multiply(TransformMatrix, new double[] { x, y });
03555             CurrentPath.LineTo(p[0], p[1]);
03556         }
03557
03558         public void MoveTo(double x, double y)
```

```
03559         {
03560             double[] p = MatrixUtils.Multiply(TransformMatrix, new double[] { x, y });
03561             CurrentPath.MoveTo(p[0], p[1]);
03562         }
03563 
03564         public void Close()
03565         {
03566             CurrentPath.Close();
03567         }
03568 
03569         public void Restore()
03570         {
03571             TransformMatrix = transformMatrices.Pop();
03572         }
03573 
03574         public void Rotate(double angle)
03575         {
03576             TransformMatrix = MatrixUtils.Rotate(TransformMatrix, angle);
03577         }
03578 
03579         public void Save()
03580         {
03581             transformMatrices.Push((double[,])TransformMatrix.Clone());
03582         }
03583 
03584         public void Scale(double scaleX, double scaleY)
03585         {
03586             TransformMatrix = MatrixUtils.Scale(TransformMatrix, scaleX, scaleY);
03587         }
03588 
03589         public void Transform(double a, double b, double c, double d, double e, double f)
03590         {
03591             double[,] transfMatrix = new double[3, 3] { { a, c, e }, { b, d, f }, { 0, 0, 1 } };
03592             TransformMatrix = MatrixUtils.Multiply(TransformMatrix, transfMatrix);
03593         }
03594 
03595         public void Translate(double x, double y)
03596         {
03597             TransformMatrix = MatrixUtils.Translate(TransformMatrix, x, y);
03598         }
03599 
03600 
03601 
03602         public double Width => throw new NotImplementedException();
03603 
03604         public double Height => throw new NotImplementedException();
03605 
03606         public Font Font { get => throw new NotImplementedException(); set => throw new
      NotImplementedException(); }
03607         public TextBaselines TextBaseline { get => throw new NotImplementedException(); set => throw
      new NotImplementedException(); }
03608 
03609         public Brush FillStyle => Colours.Black;
03610 
03611         public Brush StrokeStyle => Colours.Black;
03612 
03613         public double LineWidth { get => 1; set { } }
03614         public LineCaps LineCap { set { } }
03615         public LineJoins LineJoin { set { } }
03616         public string Tag { get => null; set { } }
03617 
03618         public void DrawRasterImage(int sourceX, int sourceY, int sourceWidth, int sourceHeight,
      double destinationX, double destinationY, double destinationWidth, double destinationHeight,
      RasterImage image)
03619         {
03620             throw new NotImplementedException();
03621         }
03622 
03623         public void Fill(FillRule fillRule)
03624         {
03625 
03626         }
03627 
03628         public void FillText(string text, double x, double y)
03629         {
03630             throw new NotImplementedException();
03631         }
03632 
03633         public void Rectangle(double x0, double y0, double width, double height)
03634         {
03635             MoveTo(x0, y0);
03636             LineTo(x0 + width, y0);
03637             LineTo(x0 + width, y0 + height);
03638             LineTo(x0, y0 + height);
03639             Close();
03640         }
03641 
```

```
03642         public void SetClippingPath()
03643         {
03644             throw new NotImplementedException();
03645         }
03646 
03647         public void SetFillStyle((int r, int g, int b, double a) style)
03648         {
03649 
03650         }
03651 
03652         public void SetFillStyle(Brush style)
03653         {
03654 
03655         }
03656 
03657         public void SetLineDash(LineDash dash)
03658         {
03659 
03660         }
03661 
03662         public void SetStrokeStyle((int r, int g, int b, double a) style)
03663         {
03664 
03665         }
03666 
03667         public void SetStrokeStyle(Brush style)
03668         {
03669 
03670         }
03671 
03672         public void Stroke()
03673         {
03674 
03675         }
03676 
03677         public void StrokeText(string text, double x, double y)
03678         {
03679             throw new NotImplementedException();
03680         }
03681 
03682         public void DrawFilteredGraphics(Graphics graphics, IFilter filter)
03683         {
03684             graphics.CopyToIGraphicsContext(this);
03685         }
03686     }
03687 }
```

8.63 Lights.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021 
00022 namespace VectSharp.ThreeD
00023 {
00024 /// <summary>
00025 /// Represents the intensity of a light source at a particular point.
00026 /// </summary>
00027     public struct LightIntensity
00028     {
00029 /// <summary>
00030 /// The intensity of the light.
00031 /// </summary>
00032         public double Intensity;
00033 
00034 /// <summary>
```

```
00035 /// The direction towards from which the light comes.
00036 /// </summary>
00037         public NormalizedVector3D Direction;
00038 
00039 /// <summary>
00040 /// Creates a new <see cref="LightIntensity"/>.
00041 /// </summary>
00042 /// <param name="intensity">The intensity of the light.</param>
00043 /// <param name="direction">The direction from which the light comes.</param>
00044         public LightIntensity(double intensity, NormalizedVector3D direction)
00045         {
00046             this.Intensity = intensity;
00047             this.Direction = direction;
00048         }
00049 
00050 /// <summary>
00051 /// Deconstructs the struct.
00052 /// </summary>
00053 /// <param name="intensity">This parameter will hold the <see cref="Intensity"/> of the light.</param>
00054 /// <param name="direction">This parameter will hold the <see cref="Direction"/> of the light.</param>
00055         public void Deconstruct(out double intensity, out NormalizedVector3D direction)
00056         {
00057             intensity = this.Intensity;
00058             direction = this.Direction;
00059         }
00060     }
00061 
00062 /// <summary>
00063 /// Represents a light source.
00064 /// </summary>
00065     public interface ILightSource
00066     {
00067 /// <summary>
00068 /// Computes the light intensity at the specified point, without taking into account any obstructions.
00069 /// </summary>
00070 /// <param name="point">The <see cref="Point3DElement"/> at which the light intensity should be
      computed.</param>
00071 /// <returns></returns>
00072         LightIntensity GetLightAt(Point3D point);
00073 
00074 /// <summary>
00075 /// Determines whether the light casts a shadow or not.
00076 /// </summary>
00077         bool CastsShadow { get; }
00078 
00079 /// <summary>
00080 /// Determines the amount of obstruction of the light that results at <paramref name="point"/> due to
      the specified <paramref name="shadowingTriangles"/>.
00081 /// </summary>
00082 /// <param name="point">The <see cref="Point3D"/> at which the obstruction should be computed.</param>
00083 /// <param name="shadowingTriangles">A collection of <see cref="Triangle3DElement"/> casting
      shadows.</param>
00084 /// <returns>1 if the light is completely obstructed, 0 if the light is completely visible, a value
      between these if the light is partially obstructed.</returns>
00085         double GetObstruction(Point3D point, IEnumerable<Triangle3DElement> shadowingTriangles);
00086     }
00087 
00088 /// <summary>
00089 /// Represents a uniform ambien light source.
00090 /// </summary>
00091     public class AmbientLightSource :  ILightSource
00092     {
00093 /// <summary>
00094 /// The intensity of the light.
00095 /// </summary>
00096         public double Intensity { get; set; }
00097 
00098 /// <inheritdoc/>
00099         public bool CastsShadow => false;
00100 
00101 /// <summary>
00102 /// Creates a new <see cref="AmbientLightSource"/> instance.
00103 /// </summary>
00104 /// <param name="intensity">The intensity of the light.</param>
00105         public AmbientLightSource(double intensity)
00106         {
00107             this.Intensity = intensity;
00108         }
00109 
00110 /// <inheritdoc/>
00111         public LightIntensity GetLightAt(Point3D point)
00112         {
00113             return new LightIntensity(Intensity, new NormalizedVector3D(double.NaN, double.NaN,
      double.NaN));
00114         }
00115 
00116 /// <inheritdoc/>
```

```
00117         public double GetObstruction(Point3D point, IEnumerable<Triangle3DElement> shadowingTriangles)
00118         {
00119             return 0;
00120         }
00121     }
00122
00123 /// <summary>
00124 /// Represents a parallel light source.
00125 /// </summary>
00126     public class ParallelLightSource :  ILightSource
00127     {
00128 /// <summary>
00129 /// The intensity of the light.
00130 /// </summary>
00131         public double Intensity { get; set; }
00132
00133 /// <summary>
00134 /// The direction along which the light travels.
00135 /// </summary>
00136         public NormalizedVector3D Direction { get; }
00137
00138 /// <summary>
00139 /// The reverse of <see cref="Direction"/>.
00140 /// </summary>
00141         public NormalizedVector3D ReverseDirection { get; }
00142
00143 /// <inheritdoc/>
00144         public bool CastsShadow { get; set; } = true;
00145
00146 /// <summary>
00147 /// Creates a new <see cref="ParallelLightSource"/> instance.
00148 /// </summary>
00149 /// <param name="intensity">The intensity of the light.</param>
00150 /// <param name="direction">The direction along which the light travels.</param>
00151         public ParallelLightSource(double intensity, NormalizedVector3D direction)
00152         {
00153             this.Intensity = intensity;
00154             this.Direction = direction;
00155             this.ReverseDirection = direction.Reverse();
00156         }
00157
00158 /// <inheritdoc/>
00159         public LightIntensity GetLightAt(Point3D point)
00160         {
00161             return new LightIntensity(Intensity, Direction);
00162         }
00163
00164 /// <inheritdoc/>
00165         public double GetObstruction(Point3D point, IEnumerable<Triangle3DElement> shadowingTriangles)
00166         {
00167             foreach (Triangle3DElement triangle in shadowingTriangles)
00168             {
00169                 Point3D? projected = triangle.ProjectOnThisPlane(point, ReverseDirection, true,
      double.PositiveInfinity);
00170
00171                 if (projected != null && Intersections3D.PointInTriangle(projected.Value,
      triangle.Point1, triangle.Point2, triangle.Point3))
00172                 {
00173                     return 1;
00174                 }
00175             }
00176
00177             return 0;
00178         }
00179     }
00180
00181 /// <summary>
00182 /// Represents a point light source.
00183 /// </summary>
00184     public class PointLightSource :  ILightSource
00185     {
00186 /// <inheritdoc/>
00187         public bool CastsShadow { get; set; } = true;
00188
00189 /// <summary>
00190 /// The position of the light source.
00191 /// </summary>
00192         public Point3D Position { get; set; }
00193
00194 /// <summary>
00195 /// The base intensity of the light.
00196 /// </summary>
00197         public double Intensity { get; set; }
00198
00199 /// <summary>
00200 /// An exponent determining how fast the light attenuates with increasing distance.  Set to 0 to
      disable distance attenuation.
```

```
00201 /// </summary>
00202         public double DistanceAttenuationExponent { get; set; } = 2;
00203 
00204 /// <summary>
00205 /// Creates a new <see cref="PointLightSource"/> instance.
00206 /// </summary>
00207 /// <param name="intensity">The intensity of the light.</param>
00208 /// <param name="position">The position of the light source.</param>
00209         public PointLightSource(double intensity, Point3D position)
00210         {
00211             this.Position = position;
00212             this.Intensity = intensity;
00213         }
00214 
00215 /// <inheritdoc/>
00216         public LightIntensity GetLightAt(Point3D point)
00217         {
00218             if (DistanceAttenuationExponent == 2)
00219             {
00220                 return new LightIntensity(Intensity / ((point.X - Position.X) * (point.X - Position.X)
      + (point.Y - Position.Y) * (point.Y - Position.Y) + (point.Z - Position.Z) * (point.Z - Position.Z)),
      (point - Position).Normalize());
00221             }
00222             else if (DistanceAttenuationExponent == 0)
00223             {
00224                 return new LightIntensity(Intensity, (point - Position).Normalize());
00225             }
00226             else
00227             {
00228                 return new LightIntensity(Intensity / Math.Pow((point.X - Position.X) * (point.X -
      Position.X) + (point.Y - Position.Y) * (point.Y - Position.Y) + (point.Z - Position.Z) * (point.Z -
      Position.Z), DistanceAttenuationExponent * 0.5), (point - Position).Normalize());
00229             }
00230         }
00231 
00232 /// <inheritdoc/>
00233         public double GetObstruction(Point3D point, IEnumerable<Triangle3DElement> shadowingTriangles)
00234         {
00235             Vector3D reverseDir = this.Position - point;
00236             double maxD = reverseDir.Modulus;
00237             NormalizedVector3D reverseDirNorm = new NormalizedVector3D(reverseDir.X / maxD,
      reverseDir.Y / maxD, reverseDir.Z / maxD, false);
00238 
00239             foreach (Triangle3DElement triangle in shadowingTriangles)
00240             {
00241                 Point3D? projected = triangle.ProjectOnThisPlane(point, reverseDirNorm, true, maxD);
00242 
00243                 if (projected != null && Intersections3D.PointInTriangle(projected.Value,
      triangle.Point1, triangle.Point2, triangle.Point3))
00244                 {
00245                     return 1;
00246                 }
00247             }
00248 
00249             return 0;
00250         }
00251     }
00252 
00253 /// <summary>
00254 /// Represents a conic spotlight.
00255 /// </summary>
00256     public class SpotlightLightSource :  ILightSource
00257     {
00258 /// <inheritdoc/>
00259         public bool CastsShadow { get; set; } = true;
00260 
00261 /// <summary>
00262 /// The position of the light source.
00263 /// </summary>
00264         public Point3D Position { get; set; }
00265 
00266 /// <summary>
00267 /// The direction of the cone axis.
00268 /// </summary>
00269         public NormalizedVector3D Direction { get; set; }
00270 
00271 /// <summary>
00272 /// The base intensity of the light.
00273 /// </summary>
00274         public double Intensity { get; set; }
00275 
00276 /// <summary>
00277 /// The angular size of the light cone, in radians.
00278 /// </summary>
00279         public double BeamWidthAngle { get; set; }
00280 
00281 /// <summary>
```

```
00282 /// The angular size of the cutoff cone, in radians.
00283 /// </summary>
00284         public double CutoffAngle { get; set; }
00285 
00286 /// <summary>
00287 /// An exponent determining how fast the light attenuates with increasing distance.  Set to 0 to
      disable distance attenuation.
00288 /// </summary>
00289         public double DistanceAttenuationExponent { get; set; } = 2;
00290 
00291 /// <summary>
00292 /// An exponent determining how fast the light attenuates between the main light cone and the cutoff
      cone.
00293 /// </summary>
00294         public double AngleAttenuationExponent { get; set; } = 1;
00295 
00296 /// <summary>
00297 /// Creates a new <see cref="SpotlightLightSource"/> instance.
00298 /// </summary>
00299 /// <param name="intensity">The intensity of the light.</param>
00300 /// <param name="position">The position of the light source.</param>
00301 /// <param name="direction">The direction of the cone's axis.</param>
00302 /// <param name="beamWidthAngle">The angular size of the light cone, in radians.</param>
00303 /// <param name="cutoffAngle">The angular size of the cutoff cone, in radians.</param>
00304         public SpotlightLightSource(double intensity, Point3D position, NormalizedVector3D direction,
      double beamWidthAngle, double cutoffAngle)
00305         {
00306             this.Position = position;
00307             this.Direction = direction;
00308             this.Intensity = intensity;
00309             this.BeamWidthAngle = beamWidthAngle;
00310             this.CutoffAngle = cutoffAngle;
00311         }
00312 
00313 /// <inheritdoc/>
00314         public LightIntensity GetLightAt(Point3D point)
00315         {
00316             double angle = Math.Atan2(((point - Position) ^ Direction).Modulus, (point - Position) *
      Direction);
00317             if (angle > Math.PI)
00318             {
00319                 angle -= 2 * Math.PI;
00320             }
00321             angle = Math.Abs(angle);
00322             double intensity;
00323 
00324             if (DistanceAttenuationExponent == 0)
00325             {
00326                 intensity = Intensity;
00327             }
00328             else if (DistanceAttenuationExponent == 2)
00329             {
00330                 intensity = Intensity / ((point.X - Position.X) * (point.X - Position.X) + (point.Y -
      Position.Y) * (point.Y - Position.Y) + (point.Z - Position.Z) * (point.Z - Position.Z));
00331             }
00332             else
00333             {
00334                 intensity = Intensity / Math.Pow((point.X - Position.X) * (point.X - Position.X) +
      (point.Y - Position.Y) * (point.Y - Position.Y) + (point.Z - Position.Z) * (point.Z - Position.Z), 0.5
      * DistanceAttenuationExponent);
00335             }
00336 
00337             if (angle <= BeamWidthAngle)
00338             {
00339                 // * 1
00340             }
00341             else if (angle >= CutoffAngle)
00342             {
00343                 intensity = 0;
00344             }
00345             else if (AngleAttenuationExponent == 0)
00346             {
00347                 // * 1
00348             }
00349             else if (AngleAttenuationExponent == 1)
00350             {
00351                 intensity *= (CutoffAngle - angle) / (CutoffAngle - BeamWidthAngle);
00352             }
00353             else
00354             {
00355                 intensity *= Math.Pow((CutoffAngle - angle) / (CutoffAngle - BeamWidthAngle),
      AngleAttenuationExponent);
00356             }
00357 
00358             return new LightIntensity(intensity, (point - Position).Normalize());
00359         }
00360 
```

```
00361 /// <inheritdoc/>
00362         public double GetObstruction(Point3D point, IEnumerable<Triangle3DElement> shadowingTriangles)
00363         {
00364             Vector3D reverseDir = this.Position - point;
00365             double maxD = reverseDir.Modulus;
00366             NormalizedVector3D reverseDirNorm = new NormalizedVector3D(reverseDir.X / maxD,
      reverseDir.Y / maxD, reverseDir.Z / maxD, false);
00367 
00368             foreach (Triangle3DElement triangle in shadowingTriangles)
00369             {
00370                 Point3D? projected = triangle.ProjectOnThisPlane(point, reverseDirNorm, true, maxD);
00371 
00372                 if (projected != null && Intersections3D.PointInTriangle(projected.Value,
      triangle.Point1, triangle.Point2, triangle.Point3))
00373                 {
00374                     return 1;
00375                 }
00376             }
00377 
00378             return 0;
00379         }
00380     }
00381 
00382 /// <summary>
00383 /// Represents a point light source with a stencil in front of it.
00384 /// </summary>
00385     public class MaskedLightSource :  ILightSource
00386     {
00387 /// <inheritdoc/>
00388         public bool CastsShadow { get; set; } = true;
00389 
00390 /// <summary>
00391 /// The position of the light source.
00392 /// </summary>
00393         public Point3D Position { get; }
00394 
00395 /// <summary>
00396 /// The projection of the <see cref="Position"/> on the mask plane along the light's <see
      cref="Direction"/>.
00397 /// </summary>
00398         public Point3D Origin { get; }
00399 
00400 /// <summary>
00401 /// The direction of the light.
00402 /// </summary>
00403         public NormalizedVector3D Direction { get; }
00404 
00405 /// <summary>
00406 /// The distance between the light source and the mask plane.
00407 /// </summary>
00408         public double Distance { get; }
00409 
00410         private List<Point3D[]> TriangulatedMask { get; }
00411 
00412 /// <summary>
00413 /// The base intensity of the light.
00414 /// </summary>
00415         public double Intensity { get; set; }
00416 
00417 /// <summary>
00418 /// An exponent determining how fast the light attenuates with increasing distance.  Set to 0 to
      disable distance attenuation.
00419 /// </summary>
00420         public double DistanceAttenuationExponent { get; set; } = 2;
00421 
00422 /// <summary>
00423 /// An exponent determining how fast the light attenuates away from the light's axis.  Set to 0 to
      disable angular attenuation.
00424 /// </summary>
00425         public double AngleAttenuationExponent { get; set; } = 1;
00426 
00427 /// <summary>
00428 /// Creates a new <see cref="MaskedLightSource"/> by triangulating the specified <see
      cref="GraphicsPath"/>.
00429 /// </summary>
00430 /// <param name="intensity">The base intensity of the light.</param>
00431 /// <param name="position">The position of the light source.</param>
00432 /// <param name="direction">The direction of the light.</param>
00433 /// <param name="distance">The distance between the light source and the mask plane.</param>
00434 /// <param name="mask">A <see cref="GraphicsPath"/> representing the transparent part of the
      mask.</param>
00435 /// <param name="maskOrientation">An angle in radians determining the orientation of the 2D mask in
      the mask plane.</param>
00436 /// <param name="triangulationResolution">The resolution to use to triangulate the <paramref
      name="mask"/>.</param>
00437         public MaskedLightSource(double intensity, Point3D position, NormalizedVector3D direction,
      double distance, GraphicsPath mask, double maskOrientation, double triangulationResolution) :
```

```
      this(intensity, position, direction, distance, mask.Triangulate(triangulationResolution, true),
      maskOrientation)
00438         {
00439
00440         }
00441
00442 /// <summary>
00443 /// Creates a new <see cref="MaskedLightSource"/> using the specified <paramref
      name="triangulatedMask"/>.
00444 /// </summary>
00445 /// <param name="intensity">The base intensity of the light.</param>
00446 /// <param name="position">The position of the light source.</param>
00447 /// <param name="direction">The direction of the light.</param>
00448 /// <param name="distance">The distance between the light source and the mask plane.</param>
00449 /// <param name="triangulatedMask">A collection of <see cref="GraphicsPath"/>s representing the
      transparent part of the mask.  Each <see cref="GraphicsPath"/> should represent a single
      triangle.</param>
00450 /// <param name="maskOrientation">An angle in radians determining the orientation of the 2D mask in
      the mask plane.</param>
00451         public MaskedLightSource(double intensity, Point3D position, NormalizedVector3D direction,
      double distance, IEnumerable<GraphicsPath> triangulatedMask, double maskOrientation)
00452         {
00453             this.Intensity = intensity;
00454             this.Position = position;
00455             this.Direction = direction;
00456             this.Distance = distance;
00457             this.Origin = this.Position + distance * this.Direction;
00458
00459             double[,] rotation = Matrix3D.RotationToAlignWithZ(direction).Inverse();
00460
00461             double[,] rotation2 = Matrix3D.RotationAroundAxis(direction, maskOrientation);
00462
00463             List<Point3D[]> maskList = new List<Point3D[]>();
00464
00465             foreach (GraphicsPath trianglePath in triangulatedMask)
00466             {
00467                 Point3D[] triangle = new Point3D[3];
00468
00469                 List<Point> points = trianglePath.GetPoints().First();
00470
00471                 for (int i = 0; i < 3; i++)
00472                 {
00473                     triangle[i] = (Point3D)((Vector3D)(rotation2 * (rotation * new
      Point3D(points[i].X, points[i].Y, 0))) + Origin);
00474                 }
00475
00476                 maskList.Add(triangle);
00477             }
00478
00479             this.TriangulatedMask = maskList;
00480         }
00481
00482 /// <inheritdoc/>
00483         public LightIntensity GetLightAt(Point3D point)
00484         {
00485             double d = Distance / ((point - this.Position) * this.Direction);
00486             Point3D pt = this.Position + (point - this.Position) * d;
00487
00488             bool contained = false;
00489
00490             foreach (Point3D[] triangle in TriangulatedMask)
00491             {
00492                 if (Intersections3D.PointInTriangle(pt, triangle[0], triangle[1], triangle[2]))
00493                 {
00494                     contained = true;
00495                     break;
00496                 }
00497             }
00498
00499             if (contained)
00500             {
00501                 double intensity;
00502
00503                 if (DistanceAttenuationExponent == 0)
00504                 {
00505                     intensity = Intensity;
00506                 }
00507                 else if (DistanceAttenuationExponent == 2)
00508                 {
00509                     intensity = Intensity / ((point.X - Position.X) * (point.X - Position.X) +
      (point.Y - Position.Y) * (point.Y - Position.Y) + (point.Z - Position.Z) * (point.Z - Position.Z));
00510                 }
00511                 else
00512                 {
00513                     intensity = Intensity / Math.Pow((point.X - Position.X) * (point.X - Position.X) +
      (point.Y - Position.Y) * (point.Y - Position.Y) + (point.Z - Position.Z) * (point.Z - Position.Z), 0.5
      * DistanceAttenuationExponent);
```

```
00514                 }
00515
00516                 if (AngleAttenuationExponent == 0)
00517                 {
00518                     // * 1
00519                 }
00520                 else
00521                 {
00522                     double angle = Math.Atan2(((point - Position) ^ Direction).Modulus, (point -
      Position) * Direction);
00523                     if (angle > Math.PI)
00524                     {
00525                         angle -= 2 * Math.PI;
00526                     }
00527
00528                     if (Math.Abs(angle) < Math.PI / 2)
00529                     {
00530                         angle = Math.Abs(angle) / Math.PI * 2;
00531
00532                         if (AngleAttenuationExponent == 1)
00533                         {
00534                             intensity *= 1 - angle;
00535                         }
00536                         else
00537                         {
00538                             intensity *= Math.Pow(1 - angle, AngleAttenuationExponent);
00539                         }
00540                     }
00541                     else
00542                     {
00543                         intensity = 0;
00544                     }
00545                 }
00546
00547                 return new LightIntensity(intensity, (point - Position).Normalize());
00548             }
00549             else
00550             {
00551                 return new LightIntensity(0, (point - Position).Normalize());
00552             }
00553         }
00554
00555 /// <inheritdoc/>
00556         public double GetObstruction(Point3D point, IEnumerable<Triangle3DElement> shadowingTriangles)
00557         {
00558             Vector3D reverseDir = this.Position - point;
00559             double maxD = reverseDir.Modulus;
00560             NormalizedVector3D reverseDirNorm = new NormalizedVector3D(reverseDir.X / maxD,
      reverseDir.Y / maxD, reverseDir.Z / maxD, false);
00561
00562             foreach (Triangle3DElement triangle in shadowingTriangles)
00563             {
00564                 Point3D? projected = triangle.ProjectOnThisPlane(point, reverseDirNorm, true, maxD);
00565
00566                 if (projected != null && Intersections3D.PointInTriangle(projected.Value,
      triangle.Point1, triangle.Point2, triangle.Point3))
00567                 {
00568                     return 1;
00569                 }
00570             }
00571
00572             return 0;
00573         }
00574     }
00575
00576 /// <summary>
00577 /// Represents a light source emitting light from a circular area.
00578 /// </summary>
00579     public class AreaLightSource :  ILightSource
00580     {
00581 /// <inheritdoc/>
00582         public bool CastsShadow { get; set; } = true;
00583
00584 /// <summary>
00585 /// The centre of the light-emitting area.
00586 /// </summary>
00587         public Point3D Center { get; }
00588
00589         private Point3D VirtualSource { get; }
00590
00591 /// <summary>
00592 /// The direction of the light's main axis, i.e.  the normal to the plane containing the
      light-emitting area.
00593 /// </summary>
00594         public NormalizedVector3D Direction { get; }
00595
00596 /// <summary>
```

```
00597 /// The radius of the light emitting area.
00598 /// </summary>
00599         public double Radius { get; }
00600
00601 /// <summary>
00602 /// The radius of the penumbra area.
00603 /// </summary>
00604         public double PenumbraRadius { get; }
00605
00606 /// <summary>
00607 /// The base intensity of the light.
00608 /// </summary>
00609         public double Intensity { get; set; }
00610
00611 /// <summary>
00612 /// The distance between the focal point of the light and the light's <see cref="Center"/>.
00613 /// </summary>
00614         public double SourceDistance { get; }
00615
00616 /// <summary>
00617 /// An exponent determining how fast the light attenuates with increasing distance.  Set to 0 to
      disable distance attenuation.
00618 /// </summary>
00619         public double DistanceAttenuationExponent { get; set; } = 2;
00620
00621 /// <summary>
00622 /// An exponent determining how fast the light attenuates between the light-emitting area radius and
      the penumbra radius.
00623 /// </summary>
00624         public double PenumbraAttenuationExponent { get; set; } = 1;
00625
00626 /// <summary>
00627 /// The number of points to use when determining the amount of light that is obstructed at a certain
      point.
00628 /// </summary>
00629         public int ShadowSamplingPointCount { get; }
00630
00631         private Point3D[] ShadowSamplingPoints { get; }
00632
00633 /// <summary>
00634 /// Creates a new <see cref="AreaLightSource"/> instance.
00635 /// </summary>
00636 /// <param name="intensity">The base intensity of the light.</param>
00637 /// <param name="center">The centre of the light-emitting area.</param>
00638 /// <param name="radius">The radius of the light-emitting area.</param>
00639 /// <param name="penumbraRadius">The radius of the penumbra area.</param>
00640 /// <param name="direction">The direction of the light.</param>
00641 /// <param name="sourceDistance">The distance between the focal point of the light and the light's
      center.</param>
00642 /// <param name="shadowSamplingPointCount">The number of points to use when determining the amount of
      light that is obstructed at a certain point.</param>
00643         public AreaLightSource(double intensity, Point3D center, double radius, double penumbraRadius,
      NormalizedVector3D direction, double sourceDistance, int shadowSamplingPointCount)
00644         {
00645             if (shadowSamplingPointCount < 1)
00646             {
00647                 throw new ArgumentOutOfRangeException(nameof(shadowSamplingPointCount),
      shadowSamplingPointCount, "The number of shadow sampling points must be greater than or equal to 1!");
00648             }
00649
00650             this.Center = center;
00651             this.Direction = direction;
00652             this.Intensity = intensity;
00653             this.Radius = radius;
00654             this.PenumbraRadius = penumbraRadius;
00655             this.SourceDistance = sourceDistance;
00656
00657             this.VirtualSource = this.Center - this.Direction * this.SourceDistance;
00658
00659             this.ShadowSamplingPointCount = shadowSamplingPointCount;
00660
00661             NormalizedVector3D yAxis;
00662
00663             if (Math.Abs(new Vector3D(0, 1, 0) * this.Direction) < 1)
00664             {
00665                 yAxis = (new Vector3D(0, 1, 0) - (new Vector3D(0, 1, 0) * this.Direction) *
      this.Direction).Normalize();
00666             }
00667             else
00668             {
00669                 yAxis = (new Vector3D(0, 0, 1) - (new Vector3D(0, 0, 1) * this.Direction) *
      this.Direction).Normalize();
00670             }
00671
00672             NormalizedVector3D xAxis = (this.Direction ^ yAxis).Normalize();
00673
00674             this.ShadowSamplingPoints = new Point3D[shadowSamplingPointCount - 1];
```

```
00675
00676             for (int i = 0; i < shadowSamplingPointCount - 1; i++)
00677             {
00678                 double r = ((double)i / (shadowSamplingPointCount - 2)) * this.Radius;
00679                 double theta = (double)i / (shadowSamplingPointCount - 2) * 2 * Math.PI *
      ((shadowSamplingPointCount - 1) / 3.7);
00680
00681                 double x = r * Math.Cos(theta);
00682                 double y = r * Math.Sin(theta);
00683
00684                 Point3D pt = this.Center + x * xAxis + y * yAxis;
00685                 this.ShadowSamplingPoints[i] = pt;
00686             }
00687         }
00688
00689 /// <inheritdoc/>
00690         public LightIntensity GetLightAt(Point3D point)
00691         {
00692             double denom = ((point - this.VirtualSource) * this.Direction);
00693
00694             if (denom <= 0)
00695             {
00696                 return new LightIntensity(0, (point - this.VirtualSource).Normalize());
00697             }
00698             else
00699             {
00700                 double d = this.SourceDistance / denom;
00701                 Point3D pt = this.VirtualSource + d * (point - this.VirtualSource);
00702                 double distSq = (pt.X - this.Center.X) * (pt.X - this.Center.X) + (pt.Y -
      this.Center.Y) * (pt.Y - this.Center.Y) + (pt.Z - this.Center.Z) * (pt.Z - this.Center.Z);
00703
00704                 if (distSq < this.Radius * this.Radius)
00705                 {
00706                     double intensity;
00707                     if (DistanceAttenuationExponent == 0)
00708                     {
00709                         intensity = Intensity;
00710                     }
00711                     else if (DistanceAttenuationExponent == 2)
00712                     {
00713                         intensity = Intensity / ((point.X - this.VirtualSource.X) * (point.X -
      this.VirtualSource.X) + (point.Y - this.VirtualSource.Y) * (point.Y - this.VirtualSource.Y) + (point.Z
      - this.VirtualSource.Z) * (point.Z - this.VirtualSource.Z));
00714                     }
00715                     else
00716                     {
00717                         intensity = Intensity / Math.Pow((point.X - this.VirtualSource.X) * (point.X -
      this.VirtualSource.X) + (point.Y - this.VirtualSource.Y) * (point.Y - this.VirtualSource.Y) + (point.Z
      - this.VirtualSource.Z) * (point.Z - this.VirtualSource.Z), 0.5 * DistanceAttenuationExponent);
00718                     }
00719
00720                     return new LightIntensity(intensity, (point - this.VirtualSource).Normalize());
00721                 }
00722                 else if (distSq < this.PenumbraRadius * this.PenumbraRadius)
00723                 {
00724                     double intensity;
00725                     if (DistanceAttenuationExponent == 0)
00726                     {
00727                         intensity = Intensity;
00728                     }
00729                     else if (DistanceAttenuationExponent == 2)
00730                     {
00731                         intensity = Intensity / ((point.X - this.VirtualSource.X) * (point.X -
      this.VirtualSource.X) + (point.Y - this.VirtualSource.Y) * (point.Y - this.VirtualSource.Y) + (point.Z
      - VirtualSource.Z) * (point.Z - this.VirtualSource.Z));
00732                     }
00733                     else
00734                     {
00735                         intensity = Intensity / Math.Pow((point.X - this.VirtualSource.X) * (point.X -
      this.VirtualSource.X) + (point.Y - this.VirtualSource.Y) * (point.Y - this.VirtualSource.Y) + (point.Z
      - this.VirtualSource.Z) * (point.Z - this.VirtualSource.Z), 0.5 * DistanceAttenuationExponent);
00736                     }
00737
00738                     if (PenumbraAttenuationExponent == 0)
00739                     {
00740                         //intensity *= 1;
00741                     }
00742                     else if (PenumbraAttenuationExponent == 1)
00743                     {
00744                         double factor = (Math.Sqrt(distSq) - this.Radius) / (this.PenumbraRadius -
      this.Radius);
00745                         intensity *= 1 - factor;
00746                     }
00747                     else
00748                     {
00749                         double factor = (Math.Sqrt(distSq) - this.Radius) / (this.PenumbraRadius -
      this.Radius);
```

```
00750                         intensity *= Math.Pow(1 - factor, PenumbraAttenuationExponent);
00751                     }
00752 
00753                     return new LightIntensity(intensity, (point - this.VirtualSource).Normalize());
00754                 }
00755                 else
00756                 {
00757                     return new LightIntensity(0, (point - this.VirtualSource).Normalize());
00758                 }
00759             }
00760         }
00761 
00762 /// <inheritdoc/>
00763         public double GetObstruction(Point3D point, IEnumerable<Triangle3DElement> shadowingTriangles)
00764         {
00765             double totalObstruction = 0;
00766             int sampleCount = 0;
00767 
00768             {
00769                 Vector3D reverseDir = this.VirtualSource - point;
00770 
00771                 double denom = (reverseDir * this.Direction);
00772 
00773                 if (denom == 0)
00774                 {
00775                     //totalObstruction += 0;
00776                     sampleCount++;
00777                 }
00778                 else
00779                 {
00780                     double d = this.SourceDistance / denom;
00781                     Point3D pt = this.VirtualSource + d * reverseDir;
00782 
00783                     double maxD = (point - pt).Modulus;
00784                     NormalizedVector3D reverseDirNorm = new NormalizedVector3D(reverseDir.X / maxD,
      reverseDir.Y / maxD, reverseDir.Z / maxD, false);
00785 
00786                     bool found = false;
00787 
00788                     foreach (Triangle3DElement triangle in shadowingTriangles)
00789                     {
00790                         Point3D? projected = triangle.ProjectOnThisPlane(point, reverseDirNorm, true,
      maxD);
00791 
00792                         if (projected != null && Intersections3D.PointInTriangle(projected.Value,
      triangle.Point1, triangle.Point2, triangle.Point3))
00793                         {
00794                             totalObstruction += 1;
00795                             sampleCount++;
00796                             found = true;
00797                             break;
00798                         }
00799                     }
00800 
00801                     if (!found)
00802                     {
00803                         //totalObstruction += 0;
00804                         sampleCount++;
00805                     }
00806                 }
00807             }
00808 
00809 
00810             for (int i = 0; i < ShadowSamplingPoints.Length; i++)
00811             {
00812                 Vector3D reverseDir = ShadowSamplingPoints[i] - point;
00813 
00814                 double maxD = reverseDir.Modulus;
00815                 NormalizedVector3D reverseDirNorm = new NormalizedVector3D(reverseDir.X / maxD,
      reverseDir.Y / maxD, reverseDir.Z / maxD, false);
00816 
00817                 bool found = false;
00818 
00819                 foreach (Triangle3DElement triangle in shadowingTriangles)
00820                 {
00821                     Point3D? projected = triangle.ProjectOnThisPlane(point, reverseDirNorm, true,
      maxD);
00822 
00823                     if (projected != null && Intersections3D.PointInTriangle(projected.Value,
      triangle.Point1, triangle.Point2, triangle.Point3))
00824                     {
00825                         totalObstruction += 1;
00826                         sampleCount++;
00827                         found = true;
00828                         break;
00829                     }
00830                 }
```

```
00831
00832                 if (!found)
00833                 {
00834                     //totalObstruction += 0;
00835                     sampleCount++;
00836                 }
00837             }
00838
00839             return totalObstruction / sampleCount;
00840         }
00841     }
00842 }
```

8.64 Materials.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Diagnostics;
00021 using System.Linq;
00022 using System.Runtime.InteropServices.ComTypes;
00023 using System.Security.Cryptography.X509Certificates;
00024 using System.Text;
00025
00026 namespace VectSharp.ThreeD
00027 {
00028 /// <summary>
00029 /// Represents a material used to the determine the appearance of <see cref="Triangle3DElement"/>.
00030 /// </summary>
00031     public interface IMaterial
00032     {
00033 /// <summary>
00034 /// Obtains the <see cref="Colour"/> at the specified point.
00035 /// </summary>
00036 /// <param name="point">The point whose colour should be determined.</param>
00037 /// <param name="surfaceNormal">The normal to the surface at the specified <paramref
      name="point"/>.</param>
00038 /// <param name="camera">The camera being used to render the scene.</param>
00039 /// <param name="lights">A list of light sources that are present in the scene.</param>
00040 /// <param name="obstructions">A list of values indicating how obstructed each light source
      is.</param>
00041 /// <returns>The <see cref="Colour"/> of the specified point.</returns>
00042         Colour GetColour(Point3D point, NormalizedVector3D surfaceNormal, Camera camera,
      IList<ILightSource> lights, IList<double> obstructions);
00043     }
00044
00045 /// <summary>
00046 /// Represents a material that always has the same colour, regardless of light.
00047 /// </summary>
00048     public class ColourMaterial :  IMaterial
00049     {
00050 /// <summary>
00051 /// The colour of the material.
00052 /// </summary>
00053         public Colour Colour { get; }
00054
00055 /// <summary>
00056 /// Creates a new <see cref="ColourMaterial"/> instance.
00057 /// </summary>
00058 /// <param name="colour">The colour of the material.</param>
00059         public ColourMaterial(Colour colour)
00060         {
00061             this.Colour = colour;
00062         }
00063
00064 /// <inheritdoc/>
00065         public Colour GetColour(Point3D point, NormalizedVector3D surfaceNormal, Camera camera,
      IList<ILightSource> lights, IList<double> obstructions)
```

```
00066         {
00067             return Colour;
00068         }
00069     }
00070 
00071 /// <summary>
00072 /// Represents a material that uses a Phong reflection model to determine the colour of the material
      based on the light sources that hit it.
00073 /// </summary>
00074     public class PhongMaterial :  IMaterial
00075     {
00076 /// <summary>
00077 /// The base colour of the material.
00078 /// </summary>
00079         public Colour Colour { get; }
00080 
00081         private (double L, double a, double b) LabColour;
00082         private (double H, double S, double L) LabBlackHSL;
00083         private (double H, double S, double L) LabWhiteHSL;
00084         private double IntensityExponent;
00085         private double TotalLength;
00086 
00087 /// <summary>
00088 /// A coefficient determining how much ambient light is reflected by the material.
00089 /// </summary>
00090         public double AmbientReflectionCoefficient { get; set; } = 1;
00091 
00092 /// <summary>
00093 /// A coefficient determining how much directional light is reflected by the material.
00094 /// </summary>
00095         public double DiffuseReflectionCoefficient { get; set; } = 1;
00096 
00097 /// <summary>
00098 /// A coefficient determining the intensity of specular highlights.
00099 /// </summary>
00100         public double SpecularReflectionCoefficient { get; set; } = 1;
00101 
00102 /// <summary>
00103 /// A coefficient determining the extent of specular highlights.
00104 /// </summary>
00105         public double SpecularShininess { get; set; } = 1;
00106 
00107 /// <summary>
00108 /// Creates a new <see cref="PhongMaterial"/> instance.
00109 /// </summary>
00110 /// <param name="colour">The base colour of the material.</param>
00111         public PhongMaterial(Colour colour)
00112         {
00113             this.Colour = colour;
00114             this.LabColour = colour.ToLab();
00115             this.LabBlackHSL = Colour.FromLab(0, LabColour.a, LabColour.b).ToHSL();
00116             this.LabWhiteHSL = Colour.FromLab(1, LabColour.a, LabColour.b).ToHSL();
00117             this.TotalLength = 1 + LabBlackHSL.L + (1 - LabWhiteHSL.L);
00118             this.IntensityExponent = Math.Log((LabBlackHSL.L + LabColour.L) / TotalLength) /
      Math.Log(0.5);
00119         }
00120 
00121         private Colour GetScaledColour(double intensity)
00122         {
00123             intensity = Math.Max(Math.Min(intensity, 1), 0);
00124 
00125             double pos = Math.Pow(intensity, IntensityExponent) * TotalLength;
00126 
00127             if (pos <= LabBlackHSL.L)
00128             {
00129                 return Colour.FromHSL(LabBlackHSL.H, LabBlackHSL.S, pos).WithAlpha(Colour.A);
00130             }
00131             else if (pos >= 1 + LabBlackHSL.L)
00132             {
00133                 return Colour.FromHSL(LabWhiteHSL.H, LabWhiteHSL.S, LabWhiteHSL.L + pos - 1 -
      LabBlackHSL.L).WithAlpha(Colour.A);
00134             }
00135             else
00136             {
00137                 return Colour.FromLab(pos - LabBlackHSL.L, LabColour.a,
      LabColour.b).WithAlpha(Colour.A);
00138             }
00139         }
00140 
00141 /// <inheritdoc/>
00142         public Colour GetColour(Point3D point, NormalizedVector3D surfaceNormal, Camera camera,
      IList<ILightSource> lights, IList<double> obstructions)
00143         {
00144             double intensity = 0;
00145 
00146             for (int i = 0; i < lights.Count; i++)
00147             {
```

```
00148                 (double lightIntensity, NormalizedVector3D lightDirection) =
      lights[i].GetLightAt(point);
00149                 lightIntensity *= 1 - obstructions[i];
00150 
00151                 if (double.IsNaN(lightDirection.X))
00152                 {
00153                     intensity += lightIntensity * AmbientReflectionCoefficient;
00154                 }
00155                 else
00156                 {
00157                     double dotProd = lightDirection * surfaceNormal;
00158 
00159                     intensity += lightIntensity * Math.Max(0, dotProd) * DiffuseReflectionCoefficient;
00160 
00161                     if (dotProd > 0)
00162                     {
00163                         NormalizedVector3D mirroredDirection = (lightDirection - 2 * dotProd *
      surfaceNormal).Normalize();
00164 
00165                         NormalizedVector3D cameraDirection = (camera.ViewPoint - point).Normalize();
00166 
00167                         double dotProd2 = mirroredDirection * cameraDirection;
00168 
00169                         if (dotProd2 >= 0)
00170                         {
00171                             intensity += lightIntensity * Math.Pow(dotProd2, SpecularShininess) *
      SpecularReflectionCoefficient;
00172                         }
00173                     }
00174                 }
00175             }
00176 
00177             return GetScaledColour(intensity);
00178         }
00179     }
00180 }
```

8.65 ObjectFactory.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021 using System.Text;
00022 
00023 namespace VectSharp.ThreeD
00024 {
00025 /// <summary>
00026 /// A static class containing methods to create complex 3D objects.
00027 /// </summary>
00028     public static class ObjectFactory
00029     {
00030 /// <summary>
00031 /// Creates a cube.
00032 /// </summary>
00033 /// <param name="center">The centre of the cube.</param>
00034 /// <param name="size">The length of each side of the cube.</param>
00035 /// <param name="fill">A collection of materials that will be applied to the <see
      cref="Triangle3DElement"/>s returned by this method.</param>
00036 /// <param name="tag">A tag that will be applied to the <see cref="Triangle3DElement"/>s returned by
      this method.</param>
00037 /// <param name="zIndex">A z-index that will be applied to the <see cref="Triangle3DElement"/>s
      returned by this method.</param>
00038 /// <returns>A list of <see cref="Triangle3DElement"/>s that constitute the cube.</returns>
00039         public static List<Element3D> CreateCube(Point3D center, double size, IEnumerable<IMaterial>
      fill, string tag = null, int zIndex = 0)
00040         {
```

```
00041             return CreateCuboid(center, size, size, size, fill, tag, zIndex);
00042         }
00043 
00044 /// <summary>
00045 /// Creates a cuboid.
00046 /// </summary>
00047 /// <param name="center">The centre of the cube.</param>
00048 /// <param name="sizeX">The length of the sides of the cube parallel to the x axis.</param>
00049 /// <param name="sizeY">The length of the sides of the cube parallel to the y axis.</param>
00050 /// <param name="sizeZ">The length of the sides of the cube parallel to the z axis.</param>
00051 /// <param name="fill">A collection of materials that will be applied to the <see
      cref="Triangle3DElement"/>s returned by this method.</param>
00052 /// <param name="tag">A tag that will be applied to the <see cref="Triangle3DElement"/>s returned by
      this method.</param>
00053 /// <param name="zIndex">A z-index that will be applied to the <see cref="Triangle3DElement"/>s
      returned by this method.</param>
00054 /// <returns>A list of <see cref="Triangle3DElement"/>s that constitute the cuboid.</returns>
00055         public static List<Element3D> CreateCuboid(Point3D center, double sizeX, double sizeY, double
      sizeZ, IEnumerable<IMaterial> fill, string tag = null, int zIndex = 0)
00056         {
00057             Point3D A = new Point3D(center.X - sizeX * 0.5, center.Y - sizeY * 0.5, center.Z - sizeZ *
      0.5);
00058             Point3D B = new Point3D(center.X + sizeX * 0.5, center.Y - sizeY * 0.5, center.Z - sizeZ *
      0.5);
00059             Point3D C = new Point3D(center.X + sizeX * 0.5, center.Y + sizeY * 0.5, center.Z - sizeZ *
      0.5);
00060             Point3D D = new Point3D(center.X - sizeX * 0.5, center.Y + sizeY * 0.5, center.Z - sizeZ *
      0.5);
00061 
00062             Point3D E = new Point3D(center.X - sizeX * 0.5, center.Y - sizeY * 0.5, center.Z + sizeZ *
      0.5);
00063             Point3D F = new Point3D(center.X + sizeX * 0.5, center.Y - sizeY * 0.5, center.Z + sizeZ *
      0.5);
00064             Point3D G = new Point3D(center.X + sizeX * 0.5, center.Y + sizeY * 0.5, center.Z + sizeZ *
      0.5);
00065             Point3D H = new Point3D(center.X - sizeX * 0.5, center.Y + sizeY * 0.5, center.Z + sizeZ *
      0.5);
00066 
00067 
00068             List<Element3D> tbr = new List<Element3D>();
00069 
00070             tbr.AddRange(CreateRectangle(A, B, C, D, fill, tag, zIndex));
00071             tbr.AddRange(CreateRectangle(H, G, F, E, fill, tag, zIndex));
00072 
00073             tbr.AddRange(CreateRectangle(E, F, B, A, fill, tag, zIndex));
00074             tbr.AddRange(CreateRectangle(G, H, D, C, fill, tag, zIndex));
00075 
00076             tbr.AddRange(CreateRectangle(A, D, H, E, fill, tag, zIndex));
00077             tbr.AddRange(CreateRectangle(F, G, C, B, fill, tag, zIndex));
00078 
00079             return tbr;
00080         }
00081 
00082 /// <summary>
00083 /// Creates a quadrilater.  All the vertices need not be coplanar.
00084 /// </summary>
00085 /// <param name="point1">The first vertex of the quadrilater.</param>
00086 /// <param name="point2">The second vertex of the quadrilater.</param>
00087 /// <param name="point3">The third vertex of the quadrilater.</param>
00088 /// <param name="point4">The fourth vertex of the quadrilater.</param>
00089 /// <param name="fill">A collection of materials that will be applied to the <see
      cref="Triangle3DElement"/>s returned by this method.</param>
00090 /// <param name="tag">A tag that will be applied to the <see cref="Triangle3DElement"/>s returned by
      this method.</param>
00091 /// <param name="zIndex">A z-index that will be applied to the <see cref="Triangle3DElement"/>s
      returned by this method.</param>
00092 /// <returns>A list containing two <see cref="Triangle3DElement"/>s representing the
      quadrilater.</returns>
00093         public static List<Element3D> CreateRectangle(Point3D point1, Point3D point2, Point3D point3,
      Point3D point4, IEnumerable<IMaterial> fill, string tag = null, int zIndex = 0)
00094         {
00095             Triangle3DElement triangle1 = new Triangle3DElement(point1, point2, point3);
00096             triangle1.Fill.AddRange(fill);
00097             triangle1.Tag = tag;
00098             triangle1.ZIndex = zIndex;
00099 
00100             Triangle3DElement triangle2 = new Triangle3DElement(point1, point3, point4);
00101             triangle2.Fill.AddRange(fill);
00102             triangle2.Tag = tag;
00103             triangle2.ZIndex = zIndex;
00104 
00105             return new List<Element3D> { triangle1, triangle2 };
00106         }
00107 
00108 /// <summary>
00109 /// Creates a quadrilater, specifying the vertex normals at the four vertices.  All the vertices need
      not be coplanar.
```

```
00110 /// </summary>
00111 /// <param name="point1">The first vertex of the quadrilater.</param>
00112 /// <param name="point2">The second vertex of the quadrilater.</param>
00113 /// <param name="point3">The third vertex of the quadrilater.</param>
00114 /// <param name="point4">The fourth vertex of the quadrilater.</param>
00115 /// <param name="point1Normal">The vertex normal at the first vertex of the quadrilater.</param>
00116 /// <param name="point2Normal">The vertex normal at the second vertex of the quadrilater.</param>
00117 /// <param name="point3Normal">The vertex normal at the third vertex of the quadrilater.</param>
00118 /// <param name="point4Normal">The vertex normal at the fourth vertex of the quadrilater.</param>
00119 /// <param name="fill">A collection of materials that will be applied to the <see
      cref="Triangle3DElement"/>s returned by this method.</param>
00120 /// <param name="tag">A tag that will be applied to the <see cref="Triangle3DElement"/>s returned by
      this method.</param>
00121 /// <param name="zIndex">A z-index that will be applied to the <see cref="Triangle3DElement"/>s
      returned by this method.</param>
00122 /// <returns>A list containing two <see cref="Triangle3DElement"/>s representing the
      quadrilater.</returns>
00123         public static List<Element3D> CreateRectangle(Point3D point1, Point3D point2, Point3D point3,
      Point3D point4, NormalizedVector3D point1Normal, NormalizedVector3D point2Normal, NormalizedVector3D
      point3Normal, NormalizedVector3D point4Normal, IEnumerable<IMaterial> fill, string tag = null, int
      zIndex = 0)
00124         {
00125             Triangle3DElement triangle1 = new Triangle3DElement(point1, point2, point3, point1Normal,
      point2Normal, point3Normal);
00126             triangle1.Fill.AddRange(fill);
00127             triangle1.Tag = tag;
00128             triangle1.ZIndex = zIndex;
00129 
00130             Triangle3DElement triangle2 = new Triangle3DElement(point1, point3, point4, point1Normal,
      point3Normal, point4Normal);
00131             triangle2.Fill.AddRange(fill);
00132             triangle2.Tag = tag;
00133             triangle2.ZIndex = zIndex;
00134 
00135             return new List<Element3D> { triangle1, triangle2 };
00136         }
00137 
00138 /// <summary>
00139 /// Creates a sphere.
00140 /// </summary>
00141 /// <param name="center">The centre of the sphere.</param>
00142 /// <param name="radius">The radius of the sphere.</param>
00143 /// <param name="steps">The number of meridians and parallels to use when generating the
      sphere.</param>
00144 /// <param name="fill">A collection of materials that will be applied to the <see
      cref="Triangle3DElement"/>s returned by this method.</param>
00145 /// <param name="tag">A tag that will be applied to the <see cref="Triangle3DElement"/>s returned by
      this method.</param>
00146 /// <param name="zIndex">A z-index that will be applied to the <see cref="Triangle3DElement"/>s
      returned by this method.</param>
00147 /// <returns>A list of <see cref="Triangle3DElement"/>s that constitute the sphere.</returns>
00148         public static List<Element3D> CreateSphere(Point3D center, double radius, int steps,
      IEnumerable<IMaterial> fill, string tag = null, int zIndex = 0)
00149         {
00150             List<Point3D> points = new List<Point3D>();
00151 
00152             for (int t = 0; t <= steps; t++)
00153             {
00154                 for (int p = 0; p < steps * 2; p++)
00155                 {
00156                     double theta = Math.PI / steps * t;
00157                     double phi = Math.PI / steps * p;
00158 
00159                     double x = center.X + radius * Math.Sin(theta) * Math.Cos(phi);
00160                     double y = center.Y + radius * Math.Sin(theta) * Math.Sin(phi);
00161                     double z = center.Z + radius * Math.Cos(theta);
00162 
00163                     points.Add(new Point3D(x, y, z));
00164 
00165                     if (t == 0 || t == steps)
00166                     {
00167                         break;
00168                     }
00169                 }
00170             }
00171 
00172             List<Element3D> tbr = new List<Element3D>(4 * steps + (points.Count - 2 - 2 * steps) * 2);
00173 
00174             for (int i = 0; i < points.Count - 1; i++)
00175             {
00176                 if (i == 0)
00177                 {
00178                     for (int j = 0; j < 2 * steps; j++)
00179                     {
00180                         Point3D p1 = points[i];
00181                         Point3D p3 = points[i + 1 + j];
00182                         Point3D p2 = points[i + 1 + (j + 1) % (2 * steps)];
```

```
00183
00184                       Triangle3DElement tri = new Triangle3DElement(p1, p2, p3, (center -
      p1).Normalize(), (center - p2).Normalize(), (center - p3).Normalize());
00185                       tri.Fill.AddRange(fill);
00186                       tri.Tag = tag;
00187                       tri.ZIndex = zIndex;
00188                       tbr.Add(tri);
00189                   }
00190               }
00191               else if (i >= points.Count - 1 - 2 * steps)
00192               {
00193                   Point3D p1 = points[i];
00194                   Point3D p3 = points[points.Count - 1];
00195                   Point3D p2 = points[points.Count - 1 - 2 * steps + (i - (points.Count - 1 - 2 *
      steps) + 1) % (2 * steps)];
00196
00197                   Triangle3DElement tri = new Triangle3DElement(p1, p2, p3, (center -
      p1).Normalize(), (center - p2).Normalize(), (center - p3).Normalize());
00198                   tri.Fill.AddRange(fill);
00199                   tri.Tag = tag;
00200                   tri.ZIndex = zIndex;
00201                   tbr.Add(tri);
00202               }
00203               else
00204               {
00205                   if ((i - 1) % (2 * steps) < 2 * steps - 1)
00206                   {
00207                       Point3D p4 = points[i + 2 * steps];
00208                       Point3D p3 = points[i + 2 * steps + 1];
00209                       Point3D p2 = points[i + 1];
00210                       Point3D p1 = points[i];
00211
00212                       tbr.AddRange(CreateRectangle(p1, p2, p3, p4, (center - p1).Normalize(),
      (center - p2).Normalize(), (center - p3).Normalize(), (center - p4).Normalize(), fill, tag, zIndex));
00213                   }
00214                   else
00215                   {
00216                       Point3D p4 = points[i + 2 * steps];
00217                       Point3D p3 = points[(i / (2 * steps)) * 2 * steps + 1];
00218                       Point3D p2 = points[(i / (2 * steps) - 1) * 2 * steps + 1];
00219                       Point3D p1 = points[i];
00220
00221                       tbr.AddRange(CreateRectangle(p1, p2, p3, p4, (center - p1).Normalize(),
      (center - p2).Normalize(), (center - p3).Normalize(), (center - p4).Normalize(), fill, tag, zIndex));
00222                   }
00223               }
00224           }
00225
00226           return tbr;
00227       }
00228
00229 /// <summary>
00230 /// Creates a tetrahedron inscribed in a sphere.
00231 /// </summary>
00232 /// <param name="center">The centre of the tetrahedron.</param>
00233 /// <param name="radius">The radius of the sphere in which the tetrahedron is inscribed.</param>
00234 /// <param name="fill">A collection of materials that will be applied to the <see
      cref="Triangle3DElement"/>s returned by this method.</param>
00235 /// <param name="tag">A tag that will be applied to the <see cref="Triangle3DElement"/>s returned by
      this method.</param>
00236 /// <param name="zIndex">A z-index that will be applied to the <see cref="Triangle3DElement"/>s
      returned by this method.</param>
00237 /// <returns>A list of <see cref="Triangle3DElement"/>s that constitute the sphere.</returns>
00238       public static List<Element3D> CreateTetrahedron(Point3D center, double radius,
      IEnumerable<IMaterial> fill, string tag = null, int zIndex = 0)
00239       {
00240           Point3D tip = new Point3D(center.X, center.Y - radius, center.Z);
00241           Point3D base1 = new Point3D(Math.Sqrt(8.0 / 9) * radius + center.X, center.Y + radius / 3,
      center.Z);
00242           Point3D base2 = new Point3D(-Math.Sqrt(2.0 / 9) * radius + center.X, center.Y + radius /
      3, center.Z + Math.Sqrt(2.0 / 3) * radius);
00243           Point3D base3 = new Point3D(-Math.Sqrt(2.0 / 9) * radius + center.X, center.Y + radius /
      3, center.Z - Math.Sqrt(2.0 / 3) * radius);
00244
00245           Triangle3DElement faceTriangle1 = new Triangle3DElement(tip, base2, base1) { Tag = tag,
      ZIndex = zIndex };
00246           faceTriangle1.Fill.AddRange(fill);
00247
00248           Triangle3DElement faceTriangle2 = new Triangle3DElement(tip, base3, base2) { Tag = tag,
      ZIndex = zIndex };
00249           faceTriangle2.Fill.AddRange(fill);
00250
00251           Triangle3DElement faceTriangle3 = new Triangle3DElement(tip, base1, base3) { Tag = tag,
      ZIndex = zIndex };
00252           faceTriangle3.Fill.AddRange(fill);
00253
00254           Triangle3DElement baseTriangle = new Triangle3DElement(base1, base2, base3) { Tag = tag,
```

```
    ZIndex = zIndex };
00255             baseTriangle.Fill.AddRange(fill);
00256
00257             return new List<Element3D>() { faceTriangle1, faceTriangle2, faceTriangle3, baseTriangle
    };
00258         }
00259
00260 /// <summary>
00261 /// Creates a flat polygon.
00262 /// </summary>
00263 /// <param name="polygon2D">A 2D <see cref="GraphicsPath"/> representing the polygon.</param>
00264 /// <param name="triangulationResolution">The resolution that will be used to linearise curve segments
    in the <see cref="GraphicsPath"/>.</param>
00265 /// <param name="origin">A <see cref="Point3D"/> that will correspond to the origin of the 2D
    reference system.</param>
00266 /// <param name="xAxis">A <see cref="NormalizedVector3D"/> that will correspond to the x axis of the
    2D reference system.  This will be orthonormalised to the <paramref name="yAxis"/>.</param>
00267 /// <param name="yAxis">A <see cref="NormalizedVector3D"/> that will correspond to the y axis of the
    2D reference system.</param>
00268 /// <param name="reverseTriangles">Indicates whether the order of the points (and thus the normals) of
    all the triangles returned by this method should be reversed.</param>
00269 /// <param name="fill">A collection of materials that will be applied to the <see
    cref="Triangle3DElement"/>s returned by this method.</param>
00270 /// <param name="tag">A tag that will be applied to the <see cref="Triangle3DElement"/>s returned by
    this method.</param>
00271 /// <param name="zIndex">A z-index that will be applied to the <see cref="Triangle3DElement"/>s
    returned by this method.</param>
00272 /// <returns>A list of <see cref="Triangle3DElement"/>s that constitute the polygon.</returns>
00273         public static List<Element3D> CreatePolygon(GraphicsPath polygon2D, double
    triangulationResolution, Point3D origin, NormalizedVector3D xAxis, NormalizedVector3D yAxis, bool
    reverseTriangles, IEnumerable<IMaterial> fill, string tag = null, int zIndex = 0)
00274         {
00275             xAxis = (xAxis - yAxis * (xAxis * yAxis)).Normalize();
00276
00277             List<GraphicsPath> triangles = polygon2D.Triangulate(triangulationResolution,
    true).ToList();
00278
00279             List<Element3D> tbr = new List<Element3D>(triangles.Count);
00280
00281             for (int i = 0; i < triangles.Count; i++)
00282             {
00283                 Point p1 = triangles[i].Segments[0].Point;
00284                 Point p2 = triangles[i].Segments[1].Point;
00285                 Point p3 = triangles[i].Segments[2].Point;
00286
00287                 Point3D p13D = origin + xAxis * p1.X + yAxis * p1.Y;
00288                 Point3D p23D = origin + xAxis * p2.X + yAxis * p2.Y;
00289                 Point3D p33D = origin + xAxis * p3.X + yAxis * p3.Y;
00290
00291                 Triangle3DElement t = !reverseTriangles ?  new Triangle3DElement(p13D, p23D, p33D) :
    new Triangle3DElement(p13D, p33D, p23D);
00292                 t.Fill.AddRange(fill);
00293                 t.Tag = tag;
00294                 t.ZIndex = zIndex;
00295                 tbr.Add(t);
00296             }
00297
00298             return tbr;
00299         }
00300
00301 /// <summary>
00302 /// Creates a prism with the specified base.
00303 /// </summary>
00304 /// <param name="polygonBase2D">A 2D <see cref="GraphicsPath"/> representing the base of the
    prism.</param>
00305 /// <param name="triangulationResolution">The resolution that will be used to linearise curve segments
    in the <see cref="GraphicsPath"/>.</param>
00306 /// <param name="bottomOrigin">A <see cref="Point3D"/> that will correspond to the origin of the 2D
    reference system of the bottom base.</param>
00307 /// <param name="topOrigin">A <see cref="Point3D"/> that will correspond to the origin of the 2D
    reference system of the top base.</param>
00308 /// <param name="baseXAxis">A <see cref="NormalizedVector3D"/> that will correspond to the x axis of
    the 2D reference system of the bases.  This will be orthonormalised to the <paramref
    name="baseYAxis"/>.</param>
00309 /// <param name="baseYAxis">A <see cref="NormalizedVector3D"/> that will correspond to the y axis of
    the 2D reference system of the bases.</param>
00310 /// <param name="fill">A collection of materials that will be applied to the <see
    cref="Triangle3DElement"/>s returned by this method.</param>
00311 /// <param name="tag">A tag that will be applied to the <see cref="Triangle3DElement"/>s returned by
    this method.</param>
00312 /// <param name="zIndex">A z-index that will be applied to the <see cref="Triangle3DElement"/>s
    returned by this method.</param>
00313 /// <returns>A list of <see cref="Triangle3DElement"/>s that constitute the prism.</returns>
00314         public static List<Element3D> CreatePrism(GraphicsPath polygonBase2D, double
    triangulationResolution, Point3D bottomOrigin, Point3D topOrigin, NormalizedVector3D baseXAxis,
    NormalizedVector3D baseYAxis, IEnumerable<IMaterial> fill, string tag = null, int zIndex = 0)
00315         {
```

```
00316             baseXAxis = (baseXAxis - baseYAxis * (baseXAxis * baseYAxis)).Normalize();
00317
00318             List<Element3D> tbr = new List<Element3D>();
00319
00320             bool orientation = (baseXAxis ^ baseYAxis) * (bottomOrigin - topOrigin) > 0;
00321
00322             double[,] matrix1 = Matrix3D.RotationToAlignAWithB(new NormalizedVector3D(0, 1, 0),
      baseYAxis);
00323             double[,] matrix2 = Matrix3D.RotationToAlignAWithB(((Vector3D)(matrix1 * new Point3D(1, 0,
      0))).Normalize(), baseXAxis);
00324
00325             List<List<NormalizedVector3D» normals = (from el2 in
      polygonBase2D.GetLinearisationPointsNormals(triangulationResolution) select (from el in el2 select
      (el.X * baseXAxis + el.Y * baseYAxis).Normalize()).ToList()).ToList();
00326
00327             polygonBase2D = polygonBase2D.Linearise(triangulationResolution);
00328
00329             tbr.AddRange(CreatePolygon(polygonBase2D, triangulationResolution, bottomOrigin,
      baseXAxis, baseYAxis, orientation, fill, tag, zIndex));
00330             tbr.AddRange(CreatePolygon(polygonBase2D, triangulationResolution, topOrigin, baseXAxis,
      baseYAxis, !orientation, fill, tag, zIndex));
00331
00332             List<List<Point3D» bottomPoints = (from el2 in polygonBase2D.GetPoints() select (from el
      in el2 select (Point3D)(el.X * baseXAxis + el.Y * baseYAxis +
      (Vector3D)bottomOrigin)).ToList()).ToList();
00333             List<List<Point3D» topPoints = (from el2 in polygonBase2D.GetPoints() select (from el in
      el2 select (Point3D)(el.X * baseXAxis + el.Y * baseYAxis + (Vector3D)topOrigin)).ToList()).ToList();
00334
00335             if (orientation)
00336             {
00337                 for (int i = 0; i < bottomPoints.Count; i++)
00338                 {
00339                     for (int j = 0; j < bottomPoints[i].Count - 1; j++)
00340                     {
00341                         tbr.AddRange(CreateRectangle(bottomPoints[i][j], bottomPoints[i][j + 1],
      topPoints[i][j + 1], topPoints[i][j], normals[i][j], normals[i][j + 1], normals[i][j + 1],
      normals[i][j], fill, tag, zIndex));
00342                     }
00343                 }
00344             }
00345             else
00346             {
00347                 for (int i = 0; i < bottomPoints.Count; i++)
00348                 {
00349                     for (int j = 0; j < bottomPoints[i].Count - 1; j++)
00350                     {
00351                         tbr.AddRange(CreateRectangle(bottomPoints[i][j], topPoints[i][j],
      topPoints[i][j + 1], bottomPoints[i][j + 1], normals[i][j], normals[i][j], normals[i][j + 1],
      normals[i][j + 1], fill, tag, zIndex));
00352                     }
00353                 }
00354             }
00355
00356             return tbr;
00357         }
00358
00359 /// <summary>
00360 /// Creates a wireframe from a collection of <see cref="Element3D"/>s.
00361 /// </summary>
00362 /// <param name="object3D">The collection of <see cref="Element3D"/>s.  <see cref="Line3DElement"/>s
      and <see cref="Point3DElement"/>s are ignored.</param>
00363 /// <param name="colour">The colour of the <see cref="Line3DElement"/>s returned by this
      method.</param>
00364 /// <param name="thickness">The thickness of the <see cref="Line3DElement"/>s returned by this
      method.</param>
00365 /// <param name="lineCap">The line cap of the <see cref="Line3DElement"/>s returned by this
      method.</param>
00366 /// <param name="lineDash">The line dash of the <see cref="Line3DElement"/>s returned by this
      method.</param>
00367 /// <param name="tag">A tag that will be applied to the <see cref="Line3DElement"/>s returned by this
      method.</param>
00368 /// <param name="zIndex">A z-index that will be applied to the <see cref="Line3DElement"/>s returned
      by this method.</param>
00369 /// <returns>A list of <see cref="Line3DElement"/>s that constitute the wireframe.</returns>
00370         public static List<Element3D> CreateWireframe(IEnumerable<Element3D> object3D, Colour colour,
      double thickness = 1, LineCaps lineCap = LineCaps.Butt, LineDash?  lineDash = null, string tag = null,
      int zIndex = 0)
00371         {
00372             List<Element3D> tbr = new List<Element3D>();
00373
00374             List<Point3D[]> addedLines = new List<Point3D[]>();
00375
00376             void addLine(Point3D p1, Point3D p2)
00377             {
00378                 for (int i = 0; i < addedLines.Count; i++)
00379                 {
00380                     if ((addedLines[i][0].Equals(p1, 1e-4) && addedLines[i][1].Equals(p2, 1e-4)) ||
```

```
    (addedLines[i][0].Equals(p2, 1e-4) && addedLines[i][1].Equals(p1, 1e-4)))
00381                     {
00382                         return;
00383                     }
00384                 }
00385 
00386                 tbr.Add(new Line3DElement(p1, p2) { Colour = colour, Thickness = thickness, LineCap =
    lineCap, LineDash = lineDash ??  LineDash.SolidLine, Tag = tag, ZIndex = zIndex });
00387                 addedLines.Add(new Point3D[] { p1, p2 });
00388             }
00389 
00390             foreach (Element3D element in object3D)
00391             {
00392                 if (element is Triangle3DElement triangle)
00393                 {
00394                     addLine(triangle.Point1, triangle.Point2);
00395                     addLine(triangle.Point2, triangle.Point3);
00396                     addLine(triangle.Point3, triangle.Point1);
00397                 }
00398             }
00399 
00400             return tbr;
00401         }
00402 
00403 /// <summary>
00404 /// Obtains a list of <see cref="Point3DElement"/> corresponding to the vertices of a list of <see
    cref="Element3D"/>s.
00405 /// </summary>
00406 /// <param name="object3D">The collection of <see cref="Element3D"/>s.  <see cref="Point3DElement"/>s
    are ignored.</param>
00407 /// <param name="colour">The colour of the <see cref="Point3DElement"/>s returned by this
    method.</param>
00408 /// <param name="diameter">The diameter of the <see cref="Point3DElement"/>s returned by this
    method.</param>
00409 /// <param name="tag">A tag that will be applied to the <see cref="Point3DElement"/>s returned by this
    method.</param>
00410 /// <param name="zIndex">A z-index that will be applied to the <see cref="Point3DElement"/>s returned
    by this method.</param>
00411 /// <returns>A list of <see cref="Point3DElement"/>s corresponding to the vertices of the <see
    cref="Element3D"/>s.</returns>
00412         public static List<Element3D> CreatePoints(IEnumerable<Element3D> object3D, Colour colour,
    double diameter = 1, string tag = null, int zIndex = 0)
00413         {
00414             List<Element3D> tbr = new List<Element3D>();
00415 
00416             List<Point3D> addedPoints = new List<Point3D>();
00417 
00418             void addPoint(Point3D p)
00419             {
00420                 for (int i = 0; i < addedPoints.Count; i++)
00421                 {
00422                     if (addedPoints[i].Equals(p, 1e-4))
00423                     {
00424                         return;
00425                     }
00426                 }
00427 
00428                 tbr.Add(new Point3DElement(p) { Colour = colour, Diameter = diameter, Tag = tag,
    ZIndex = zIndex });
00429                 addedPoints.Add(p);
00430             }
00431 
00432             foreach (Element3D element in object3D)
00433             {
00434                 if (!(element is Point3DElement))
00435                 {
00436                     foreach (Point3D p in element)
00437                     {
00438                         addPoint(p);
00439                     }
00440                 }
00441             }
00442 
00443             return tbr;
00444         }
00445     }
00446 }
```

8.66 Scene.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
```

```
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020
00021 namespace VectSharp.ThreeD
00022 {
00023 /// <summary>
00024 /// Represents a 3D scene.
00025 /// </summary>
00026     public interface IScene
00027     {
00028 /// <summary>
00029 /// The <see cref="Element3D"/>s constituting the scene.
00030 /// </summary>
00031         IEnumerable<Element3D> SceneElements { get; }
00032
00033 /// <summary>
00034 /// Adds the specified <paramref name="element"/> to the scene.
00035 /// </summary>
00036 /// <param name="element">The <see cref="Element3D"/> to add.</param>
00037         void AddElement(Element3D element);
00038
00039 /// <summary>
00040 /// Adds the specified <paramref name="elements"/> to the scene.
00041 /// </summary>
00042 /// <param name="elements">A collection of <see cref="Element3D"/>s to add.</param>
00043         void AddRange(IEnumerable<Element3D> elements);
00044
00045 /// <summary>
00046 /// Replaces each element in the scene with the element returned by the <paramref
      name="replacementFunction"/>.
00047 /// </summary>
00048 /// <param name="replacementFunction">A function replacing each <see cref="Element3D"/> in the scene
      with another <see cref="Element3D"/>.</param>
00049         void Replace(Func<Element3D, Element3D> replacementFunction);
00050
00051 /// <summary>
00052 /// Replaces each element in the scene with the element(s) returned by the <paramref
      name="replacementFunction"/>.
00053 /// </summary>
00054 /// <param name="replacementFunction">A function replacing each <see cref="Element3D"/> in the scene
      with 0 or more <see cref="Element3D"/>s.</param>
00055         void Replace(Func<Element3D, IEnumerable<Element3D>» replacementFunction);
00056
00057 /// <summary>
00058 /// An object used to synchronise multithreaded rendering of the same scene.
00059 /// </summary>
00060         object SceneLock { get; }
00061     }
00062
00063 /// <summary>
00064 /// Represents a 3D scene.
00065 /// </summary>
00066     public class Scene :  IScene
00067     {
00068 /// <inheritdoc/>
00069         public object SceneLock { get; }
00070
00071         private List<Element3D> sceneElements;
00072
00073 /// <inheritdoc/>
00074         public IEnumerable<Element3D> SceneElements => sceneElements;
00075
00076 /// <summary>
00077 /// Creates a new <see cref="Scene"/>.
00078 /// </summary>
00079         public Scene()
00080         {
00081             sceneElements = new List<Element3D>();
00082             this.SceneLock = new object();
00083         }
00084
00085 /// <inheritdoc/>
00086         public void AddElement(Element3D element)
```

```
00087         {
00088             this.sceneElements.Add(element);
00089         }
00090
00091 /// <inheritdoc/>
00092         public void AddRange(IEnumerable<Element3D> elements)
00093         {
00094             this.sceneElements.AddRange(elements);
00095         }
00096
00097 /// <inheritdoc/>
00098         public void Replace(Func<Element3D, Element3D> replacementFunction)
00099         {
00100             for (int i = 0; i < sceneElements.Count; i++)
00101             {
00102                 sceneElements[i] = replacementFunction(sceneElements[i]);
00103             }
00104         }
00105
00106 /// <inheritdoc/>
00107         public void Replace(Func<Element3D, IEnumerable<Element3D>> replacementFunction)
00108         {
00109             List<Element3D> newElements = new List<Element3D>(sceneElements.Count);
00110
00111             for (int i = 0; i < this.sceneElements.Count; i++)
00112             {
00113                 newElements.AddRange(replacementFunction(sceneElements[i]));
00114             }
00115
00116             newElements.TrimExcess();
00117
00118             this.sceneElements = newElements;
00119         }
00120     }
00121 }
```

8.67 Animation.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2022 Giorgio Bianchini, University of Bristol
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Collections.Immutable;
00021 using System.Diagnostics.Contracts;
00022 using System.Linq;
00023 using VectSharp.Filters;
00024
00025 namespace VectSharp
00026 {
00027     internal interface IAnimation
00028     {
00029         IGraphicsAction StartValue { get; }
00030         IGraphicsAction EndValue { get; }
00031         IGraphicsAction Interpolate(double position, Dictionary<string, IEasing> easings);
00032     }
00033
00034     internal abstract class IAnimation<T> :  IAnimation where T : IGraphicsAction
00035     {
00036         public abstract T StartValue { get; }
00037         public abstract T EndValue { get; }
00038
00039         IGraphicsAction IAnimation.StartValue => (IGraphicsAction)StartValue;
00040
00041         IGraphicsAction IAnimation.EndValue => (IGraphicsAction)EndValue;
00042
00043         public abstract T Interpolate(double position, Dictionary<string, IEasing> easings);
00044
00045         IGraphicsAction IAnimation.Interpolate(double position, Dictionary<string, IEasing> easings)
```

```
00046         {
00047             return (IGraphicsAction)Interpolate(position, easings);
00048         }
00049     }
00050 
00051     internal class ConstantAnimation<T> :  IAnimation<T> where T : IGraphicsAction
00052     {
00053         public override T StartValue { get; }
00054 
00055         public override T EndValue => StartValue;
00056 
00057         public ConstantAnimation(T action)
00058         {
00059             this.StartValue = action;
00060         }
00061 
00062         public override T Interpolate(double position, Dictionary<string, IEasing> easings)
00063         {
00064             return this.StartValue;
00065         }
00066     }
00067 
00068     internal class StateAnimation :  IAnimation<StateAction>
00069     {
00070         public override StateAction StartValue { get; }
00071 
00072         public override StateAction EndValue { get; }
00073 
00074         public StateAnimation(StateAction startValue, StateAction endValue)
00075         {
00076             this.StartValue = startValue;
00077             this.EndValue = endValue;
00078         }
00079 
00080         public override StateAction Interpolate(double position, Dictionary<string, IEasing> easings)
00081         {
00082             if (easings != null && !string.IsNullOrEmpty(StartValue.Tag) &&
      easings.TryGetValue(StartValue.Tag, out IEasing easing))
00083             {
00084                 position = easing.Ease(position);
00085             }
00086 
00087             if (position < 0.5)
00088             {
00089                 return StartValue;
00090             }
00091             else
00092             {
00093                 return EndValue;
00094             }
00095         }
00096     }
00097 
00098     internal static class InterpolationUtils
00099     {
00100         public static Point InterpolatePoint(Point start, Point end, double position)
00101         {
00102             return new Point(start.X * (1 - position) + end.X * position, start.Y * (1 - position) +
      end.Y * position);
00103         }
00104 
00105         public static Point InterpolatePoint(Point?  start, Point?  end, double position)
00106         {
00107             return InterpolatePoint(start ??  new Point(), end ??  new Point(), position);
00108         }
00109 
00110         public static Size InterpolateSize(Size start, Size end, double position)
00111         {
00112             return new Size(start.Width * (1 - position) + end.Width * position, start.Height * (1 -
      position) + end.Height * position);
00113         }
00114 
00115         public static Size InterpolateSize(Size?  start, Size?  end, double position)
00116         {
00117             return InterpolateSize(start ??  new Size(), end ??  new Size(), position);
00118         }
00119 
00120         public static double InterpolateDouble(double start, double end, double position)
00121         {
00122             return start * (1 - position) + end * position;
00123         }
00124 
00125         public static double InterpolateDouble(double?  start, double?  end, double position)
00126         {
00127             return start * (1 - position) + end * position ??  0;
00128         }
00129 
```

```
00130         public static Colour InterpolateColour(Colour start, Colour end, double position)
00131         {
00132             return Colour.FromRgba(InterpolateDouble(start.R, end.R, position),
00133                     InterpolateDouble(start.G, end.G, position),
00134                     InterpolateDouble(start.B, end.B, position),
00135                     InterpolateDouble(start.A, end.A, position));
00136         }
00137 
00138         public static LineDash InterpolateLineDash(LineDash start, LineDash end, double position)
00139         {
00140             double[] startDashArray = start.DashArray != null && start.DashArray.Length > 0 ? start.DashArray :  new double[] { 0, 0 };
00141             double[] endDashArray = end.DashArray != null && end.DashArray.Length > 0 ?  end.DashArray :  new double[] { 0, 0 }; ;
00142 
00143             int targetLength;
00144 
00145             if (Math.Max(startDashArray.Length, endDashArray.Length) % Math.Min(startDashArray.Length, endDashArray.Length) == 0)
00146             {
00147                 targetLength = Math.Max(startDashArray.Length, endDashArray.Length);
00148             }
00149             else
00150             {
00151                 targetLength = startDashArray.Length * endDashArray.Length;
00152             }
00153 
00154             double[] targetArray = new double[targetLength];
00155 
00156             for (int i = 0; i < targetLength; i++)
00157             {
00158                 double startI = startDashArray[i % startDashArray.Length];
00159                 double endI = endDashArray[i % endDashArray.Length];
00160                 targetArray[i] = InterpolateDouble(startI, endI, position);
00161             }
00162 
00163             return new LineDash(targetArray, InterpolateDouble(start.Phase, end.Phase, position));
00164         }
00165 
00166         private static HashSet<double> GetAnchors(Brush start, Brush end)
00167         {
00168             HashSet<double> anchors = new HashSet<double>();
00169 
00170             if (start is GradientBrush gradientStart)
00171             {
00172                 for (int i = 0; i < gradientStart.GradientStops.Count; i++)
00173                 {
00174                     anchors.Add(gradientStart.GradientStops[i].Offset);
00175                 }
00176             }
00177 
00178             if (end is GradientBrush gradientEnd)
00179             {
00180                 for (int i = 0; i < gradientEnd.GradientStops.Count; i++)
00181                 {
00182                     anchors.Add(gradientEnd.GradientStops[i].Offset);
00183                 }
00184             }
00185 
00186             return anchors;
00187         }
00188 
00189         private static void AddGradientStop(List<GradientStop> stops)
00190         {
00191             int minIndex = -1;
00192             double maxInterval = double.MinValue;
00193 
00194             for (int i = 1; i < stops.Count; i++)
00195             {
00196                 double interval = stops[i].Offset - stops[i - 1].Offset;
00197 
00198                 if (interval > maxInterval)
00199                 {
00200                     maxInterval = interval;
00201                     minIndex = i;
00202                 }
00203             }
00204 
00205             if (minIndex > 0)
00206             {
00207                 stops.Insert(minIndex, new GradientStop(Colour.FromRgba((stops[minIndex - 1].Colour.R + stops[minIndex].Colour.R) * 0.5, (stops[minIndex - 1].Colour.G + stops[minIndex].Colour.G) * 0.5, (stops[minIndex - 1].Colour.B + stops[minIndex].Colour.B) * 0.5, (stops[minIndex - 1].Colour.A + stops[minIndex].Colour.A) * 0.5), stops[minIndex - 1].Offset + maxInterval * 0.5));
00208             }
00209         }
00210 
```

```
00211         public static Brush InterpolateBrush(Brush start, Brush end, double position)
00212         {
00213             if (position <= 0)
00214             {
00215                 return start;
00216             }
00217             else if (position >= 1)
00218             {
00219                 return end;
00220             }
00221             else
00222             {
00223                 HashSet<double> anchors = GetAnchors(start, end);
00224 
00225                 if (start is SolidColourBrush solidStart && end is SolidColourBrush solidEnd)
00226                 {
00227                     return new SolidColourBrush(InterpolateColour(solidStart.Colour, solidEnd.Colour,
      position));
00228                 }
00229                 else if (start is LinearGradientBrush linearStart && end is LinearGradientBrush
      linearEnd)
00230                 {
00231                     List<GradientStop> startStops = linearStart.GradientStops.ToList();
00232                     List<GradientStop> endStops = linearEnd.GradientStops.ToList();
00233 
00234                     while (startStops.Count < endStops.Count)
00235                     {
00236                         AddGradientStop(startStops);
00237                     }
00238 
00239                     while (startStops.Count > endStops.Count)
00240                     {
00241                         AddGradientStop(endStops);
00242                     }
00243 
00244                     List<GradientStop> interpolatedStops = new List<GradientStop>();
00245 
00246                     for (int i = 0; i < startStops.Count; i++)
00247                     {
00248                         interpolatedStops.Add(new GradientStop(InterpolateColour(startStops[i].Colour,
      endStops[i].Colour, position), InterpolateDouble(startStops[i].Offset, endStops[i].Offset,
      position)));
00249                     }
00250 
00251                     return new LinearGradientBrush(InterpolatePoint(linearStart.StartPoint,
      linearEnd.StartPoint, position), InterpolatePoint(linearStart.EndPoint, linearEnd.EndPoint, position),
      interpolatedStops);
00252                 }
00253                 else if (start is SolidColourBrush solidStart2 && end is LinearGradientBrush
      linearEnd2)
00254                 {
00255                     List<GradientStop> gradientStops = new List<GradientStop>(anchors.Count);
00256 
00257                     foreach (double anchor in anchors)
00258                     {
00259                         Colour startColour = solidStart2.Colour;
00260                         Colour endColour = linearEnd2.GradientStops.GetColourAt(anchor);
00261 
00262                         gradientStops.Add(new GradientStop(InterpolateColour(startColour, endColour,
      position), anchor));
00263                     }
00264 
00265                     return new LinearGradientBrush(linearEnd2.StartPoint, linearEnd2.EndPoint,
      gradientStops);
00266                 }
00267                 else if (start is LinearGradientBrush linearStart2 && end is SolidColourBrush
      solidEnd2)
00268                 {
00269                     List<GradientStop> gradientStops = new List<GradientStop>(anchors.Count);
00270 
00271                     foreach (double anchor in anchors)
00272                     {
00273                         Colour startColour = linearStart2.GradientStops.GetColourAt(anchor);
00274                         Colour endColour = solidEnd2.Colour;
00275 
00276                         gradientStops.Add(new GradientStop(InterpolateColour(startColour, endColour,
      position), anchor));
00277                     }
00278 
00279                     return new LinearGradientBrush(linearStart2.StartPoint, linearStart2.EndPoint,
      gradientStops);
00280                 }
00281                 else if (start is RadialGradientBrush radialStart && end is RadialGradientBrush
      radialEnd)
00282                 {
00283                     List<GradientStop> startStops = radialStart.GradientStops.ToList();
00284                     List<GradientStop> endStops = radialEnd.GradientStops.ToList();
```

```
00285
00286                     while (startStops.Count < endStops.Count)
00287                     {
00288                         AddGradientStop(startStops);
00289                     }
00290
00291                     while (startStops.Count > endStops.Count)
00292                     {
00293                         AddGradientStop(endStops);
00294                     }
00295
00296                     List<GradientStop> interpolatedStops = new List<GradientStop>();
00297
00298                     for (int i = 0; i < startStops.Count; i++)
00299                     {
00300                         interpolatedStops.Add(new GradientStop(InterpolateColour(startStops[i].Colour,
      endStops[i].Colour, position), InterpolateDouble(startStops[i].Offset, endStops[i].Offset,
      position)));
00301                     }
00302
00303                     return new RadialGradientBrush(InterpolatePoint(radialStart.FocalPoint,
      radialEnd.FocalPoint, position), InterpolatePoint(radialStart.Centre, radialEnd.Centre, position),
      InterpolateDouble(radialStart.Radius, radialEnd.Radius, position), interpolatedStops);
00304                 }
00305                 else if (start is SolidColourBrush solidStart3 && end is RadialGradientBrush
      radialEnd2)
00306                 {
00307                     List<GradientStop> gradientStops = new List<GradientStop>(anchors.Count);
00308
00309                     foreach (double anchor in anchors)
00310                     {
00311                         Colour startColour = solidStart3.Colour;
00312                         Colour endColour = radialEnd2.GradientStops.GetColourAt(anchor);
00313
00314                         gradientStops.Add(new GradientStop(InterpolateColour(startColour, endColour,
      position), anchor));
00315                     }
00316
00317                     return new RadialGradientBrush(radialEnd2.FocalPoint, radialEnd2.Centre,
      radialEnd2.Radius, gradientStops);
00318                 }
00319                 else if (start is RadialGradientBrush radialStart2 && end is SolidColourBrush
      solidEnd3)
00320                 {
00321                     List<GradientStop> gradientStops = new List<GradientStop>(anchors.Count);
00322
00323                     foreach (double anchor in anchors)
00324                     {
00325                         Colour startColour = radialStart2.GradientStops.GetColourAt(anchor);
00326                         Colour endColour = solidEnd3.Colour;
00327
00328                         gradientStops.Add(new GradientStop(InterpolateColour(startColour, endColour,
      position), anchor));
00329                     }
00330
00331                     return new RadialGradientBrush(radialStart2.FocalPoint, radialStart2.Centre,
      radialStart2.Radius, gradientStops);
00332                 }
00333                 else if (start is LinearGradientBrush linearStart3 && end is RadialGradientBrush
      radialEnd3)
00334                 {
00335                     Point linearVector = new Point(linearStart3.EndPoint.X -
      linearStart3.StartPoint.X, linearStart3.EndPoint.Y - linearStart3.StartPoint.Y);
00336
00337                     double modulus = linearVector.Modulus();
00338                     linearVector = new Point(linearVector.X / modulus, linearVector.Y / modulus);
00339
00340                     double radius = radialEnd3.Radius / position;// radialEnd3.Radius * (1 + 100 * (1
      - position));
00341
00342                     Point target = new Point(linearStart3.EndPoint.X - linearVector.X * radius,
      linearStart3.EndPoint.Y - linearVector.Y * radius);
00343
00344                     List<GradientStop> gradientStops = new List<GradientStop>(anchors.Count);
00345
00346                     double lOverR = (modulus + (radialEnd3.Radius - modulus) * position) /
      radialEnd3.Radius;
00347
00348                     foreach (double anchor in anchors)
00349                     {
00350                         double offset = (1 - lOverR * position) + anchor * position * lOverR;
00351
00352                         Colour startColour = linearStart3.GradientStops.GetColourAt(anchor);
00353                         Colour endColour = radialEnd3.GradientStops.GetColourAt(anchor);
00354                         gradientStops.Add(new GradientStop(InterpolateColour(startColour, endColour,
      position), offset));
00355                     }
```

```
00356
00357                   return new RadialGradientBrush(InterpolatePoint(target, radialEnd3.FocalPoint,
      position), InterpolatePoint(target, radialEnd3.Centre, position), InterpolateDouble(radius,
      radialEnd3.Radius, position), gradientStops);
00358               }
00359               else if (start is RadialGradientBrush && end is LinearGradientBrush)
00360               {
00361                   return InterpolateBrush(end, start, 1 - position);
00362               }
00363               else
00364               {
00365                   if (position < 0.5)
00366                   {
00367                       return start;
00368                   }
00369                   else
00370                   {
00371                       return end;
00372                   }
00373               }
00374           }
00375       }
00376
00377       public static int InterpolateInt(int start, int end, double position)
00378       {
00379           return (int)Math.Round(start * (1 - position) + end * position);
00380       }
00381
00382       public static Font InterpolateFont(Font start, Font end, double position)
00383       {
00384           Font tbr = new Font(position < 0.5 ?  start.FontFamily :  end.FontFamily,
      InterpolateDouble(start.FontSize, end.FontSize, position), start.Underline != null || end.Underline !=
      null);
00385
00386           if (start.Underline != null && end.Underline == null)
00387           {
00388               tbr.Underline.FollowItalicAngle = start.Underline.FollowItalicAngle;
00389               tbr.Underline.LineCap = start.Underline.LineCap;
00390               tbr.Underline.Position = start.Underline.Position;
00391               tbr.Underline.SkipDescenders = start.Underline.SkipDescenders;
00392               tbr.Underline.Thickness = InterpolateDouble(start.Underline.Thickness, 0, position);
00393           }
00394           else if (start.Underline == null && end.Underline != null)
00395           {
00396               tbr.Underline.FollowItalicAngle = end.Underline.FollowItalicAngle;
00397               tbr.Underline.LineCap = end.Underline.LineCap;
00398               tbr.Underline.Position = end.Underline.Position;
00399               tbr.Underline.SkipDescenders = end.Underline.SkipDescenders;
00400               tbr.Underline.Thickness = InterpolateDouble(0, end.Underline.Thickness, position);
00401           }
00402           else if (start.Underline != null && end.Underline != null)
00403           {
00404               tbr.Underline.FollowItalicAngle = position < 0.5 ?  start.Underline.FollowItalicAngle
      :  end.Underline.FollowItalicAngle;
00405               tbr.Underline.LineCap = position < 0.5 ?  start.Underline.LineCap :
      end.Underline.LineCap;
00406               tbr.Underline.Position = InterpolateDouble(start.Underline.Position,
      end.Underline.Position, position);
00407               tbr.Underline.SkipDescenders = position < 0.5 ?  start.Underline.SkipDescenders :
      end.Underline.SkipDescenders;
00408               tbr.Underline.Thickness = InterpolateDouble(start.Underline.Thickness, 0, position);
00409           }
00410
00411           return tbr;
00412       }
00413   }
00414
00415   internal class TransformAnimation :  IAnimation<TransformAction>
00416   {
00417       public override TransformAction StartValue { get; }
00418
00419       public override TransformAction EndValue { get; }
00420
00421       public TransformAnimation(TransformAction startValue, TransformAction endValue)
00422       {
00423           this.StartValue = startValue;
00424           this.EndValue = endValue;
00425       }
00426       public override TransformAction Interpolate(double position, Dictionary<string, IEasing>
      easings)
00427       {
00428           if (easings != null && !string.IsNullOrEmpty(StartValue.Tag) &&
      easings.TryGetValue(StartValue.Tag, out IEasing easing))
00429           {
00430               position = easing.Ease(position);
00431           }
00432
```

```
00433             if (position <= 0)
00434             {
00435                 return this.StartValue;
00436             }
00437             else if (position >= 1)
00438             {
00439                 return this.EndValue;
00440             }
00441             else
00442             {
00443                 if (this.StartValue.Delta != null && this.EndValue.Delta != null)
00444                 {
00445                     return new
      TransformAction(InterpolationUtils.InterpolatePoint(this.StartValue.Delta, this.EndValue.Delta,
      position), position < 0.5 ?  this.StartValue.Tag :  this.EndValue.Tag);
00446                 }
00447                 else if (this.StartValue.Scale != null && this.EndValue.Scale != null)
00448                 {
00449                     return new
      TransformAction(InterpolationUtils.InterpolateSize(this.StartValue.Scale, this.EndValue.Scale,
      position), position < 0.5 ?  this.StartValue.Tag :  this.EndValue.Tag);
00450                 }
00451                 else if (this.StartValue.Angle != null && this.EndValue.Angle != null)
00452                 {
00453                     return new
      TransformAction(InterpolationUtils.InterpolateDouble(this.StartValue.Angle, this.EndValue.Angle,
      position), position < 0.5 ?  this.StartValue.Tag :  this.EndValue.Tag);
00454                 }
00455                 else
00456                 {
00457                     double[,] startMatrix = this.StartValue.GetMatrix();
00458                     double[,] endMatrix = this.EndValue.GetMatrix();
00459 
00460                     double[,] tbrMatrix = new double[startMatrix.GetLength(0),
      startMatrix.GetLength(1)];
00461 
00462                     for (int i = 0; i < startMatrix.GetLength(0); i++)
00463                     {
00464                         for (int j = 0; j < startMatrix.GetLength(1); j++)
00465                         {
00466                             tbrMatrix[i, j] = startMatrix[i, j] * (1 - position) + endMatrix[i, j] *
      position;
00467                         }
00468                     }
00469 
00470                     return new TransformAction(tbrMatrix, position < 0.5 ?  this.StartValue.Tag :
      this.EndValue.Tag);
00471                 }
00472             }
00473         }
00474     }
00475 
00476     internal abstract class PrintableActionAnimation<T> :  IAnimation<T> where T : IPrintableAction,
      IGraphicsAction
00477     {
00478         public override T StartValue { get; }
00479         public override T EndValue { get; }
00480 
00481         protected abstract T InterpolateConcrete((double, LineJoins, LineCaps, LineDash, Brush, Brush)
      commonElements, double position, Dictionary<string, IEasing> easings);
00482 
00483         public PrintableActionAnimation(T startValue, T endValue)
00484         {
00485             this.StartValue = startValue;
00486             this.EndValue = endValue;
00487         }
00488 
00489         public override T Interpolate(double position, Dictionary<string, IEasing> easings)
00490         {
00491             string tag = (StartValue as IPrintableAction)?.Tag ??  (StartValue as
      IGraphicsAction)?.Tag;
00492 
00493             if (easings != null && !string.IsNullOrEmpty(tag) && easings.TryGetValue(tag, out IEasing
      easing))
00494             {
00495                 position = easing.Ease(position);
00496             }
00497 
00498             if (position <= 0)
00499             {
00500                 return StartValue;
00501             }
00502             else if (position >= 1)
00503             {
00504                 return EndValue;
00505             }
00506             else
```

```
00507             {
00508                 T tbr = InterpolateConcrete(InterpolateCommon(position), position, easings);
00509                 ((IGraphicsAction)tbr).Tag = ((IGraphicsAction)this.StartValue).Tag;
00510                 return tbr;
00511             }
00512         }
00513 
00514         protected (double, LineJoins, LineCaps, LineDash, Brush, Brush) InterpolateCommon(double
      position)
00515         {
00516             double lineWidth = InterpolationUtils.InterpolateDouble(this.StartValue.LineWidth,
      this.EndValue.LineWidth, position);
00517             LineJoins lineJoin = position < 0.5 ?  this.StartValue.LineJoin :  this.EndValue.LineJoin;
00518             LineCaps lineCap = position < 0.5 ?  this.StartValue.LineCap :  this.EndValue.LineCap;
00519             LineDash lineDash = InterpolationUtils.InterpolateLineDash(this.StartValue.LineDash,
      this.EndValue.LineDash, position);
00520 
00521             Brush stroke = null;
00522 
00523             if (this.StartValue.Stroke != null && this.EndValue.Stroke != null)
00524             {
00525                 stroke = InterpolationUtils.InterpolateBrush(this.StartValue.Stroke,
      this.EndValue.Stroke, position);
00526             }
00527 
00528             Brush fill = null;
00529 
00530             if (this.StartValue.Fill != null && this.EndValue.Fill != null)
00531             {
00532                 fill = InterpolationUtils.InterpolateBrush(this.StartValue.Fill, this.EndValue.Fill,
      position);
00533             }
00534 
00535             return (lineWidth, lineJoin, lineCap, lineDash, stroke, fill);
00536         }
00537     }
00538 
00539     internal class RectangleActionAnimation :  PrintableActionAnimation<RectangleAction>
00540     {
00541         public RectangleActionAnimation(RectangleAction startValue, RectangleAction endValue) :
      base(startValue, endValue) { }
00542 
00543         protected override RectangleAction InterpolateConcrete((double, LineJoins, LineCaps, LineDash,
      Brush, Brush) commonElements, double position, Dictionary<string, IEasing> easings)
00544         {
00545             (double lineWidth, LineJoins lineJoin, LineCaps lineCap, LineDash lineDash, Brush stroke,
      Brush fill) = commonElements;
00546 
00547             return new RectangleAction(InterpolationUtils.InterpolatePoint(this.StartValue.TopLeft,
      this.EndValue.TopLeft, position),
00548                         InterpolationUtils.InterpolateSize(this.StartValue.Size, this.EndValue.Size,
      position),
00549                         fill, stroke, lineWidth, lineCap, lineJoin, lineDash, position < 0.5 ?
      this.StartValue.Tag :  this.EndValue.Tag);
00550         }
00551     }
00552 
00553     internal class RasterImageActionAnimation :  PrintableActionAnimation<RasterImageAction>
00554     {
00555         public RasterImageActionAnimation(RasterImageAction startValue, RasterImageAction endValue) :
      base(startValue, endValue) { }
00556 
00557         protected override RasterImageAction InterpolateConcrete((double, LineJoins, LineCaps,
      LineDash, Brush, Brush) commonElements, double position, Dictionary<string, IEasing> easings)
00558         {
00559             return new RasterImageAction(InterpolationUtils.InterpolateInt(this.StartValue.SourceX,
      this.EndValue.SourceX, position),
00560                 InterpolationUtils.InterpolateInt(this.StartValue.SourceY, this.EndValue.SourceY,
      position),
00561                 InterpolationUtils.InterpolateInt(this.StartValue.SourceWidth,
      this.EndValue.SourceWidth, position),
00562                 InterpolationUtils.InterpolateInt(this.StartValue.SourceHeight,
      this.EndValue.SourceHeight, position),
00563                 InterpolationUtils.InterpolateDouble(this.StartValue.DestinationX,
      this.EndValue.DestinationX, position),
00564                 InterpolationUtils.InterpolateDouble(this.StartValue.DestinationY,
      this.EndValue.DestinationY, position),
00565                 InterpolationUtils.InterpolateDouble(this.StartValue.DestinationWidth,
      this.EndValue.DestinationWidth, position),
00566                 InterpolationUtils.InterpolateDouble(this.StartValue.DestinationHeight,
      this.EndValue.DestinationHeight, position), position < 0.5 ?  this.StartValue.Image :
      this.EndValue.Image
00567                 , position < 0.5 ?  this.StartValue.Tag :  this.EndValue.Tag);
00568         }
00569     }
00570 
00571     internal class TextActionAnimation :  PrintableActionAnimation<TextAction>
```

```
00572     {
00573         public TextActionAnimation(TextAction startValue, TextAction endValue) :  base(startValue,
      endValue) { }
00574 
00575         protected override TextAction InterpolateConcrete((double, LineJoins, LineCaps, LineDash,
      Brush, Brush) commonElements, double position, Dictionary<string, IEasing> easings)
00576         {
00577             (double lineWidth, LineJoins lineJoin, LineCaps lineCap, LineDash lineDash, Brush stroke,
      Brush fill) = commonElements;
00578 
00579             return new TextAction(InterpolationUtils.InterpolatePoint(this.StartValue.Origin,
      this.EndValue.Origin, position),
00580                 position < 0.5 ?  this.StartValue.Text :  this.EndValue.Text,
00581                 InterpolationUtils.InterpolateFont(this.StartValue.Font, this.EndValue.Font,
      position),
00582                 position < 0.5 ?  this.StartValue.TextBaseline :  this.EndValue.TextBaseline,
00583                 fill, stroke, lineWidth, lineCap, lineJoin, lineDash, position < 0.5 ?
      this.StartValue.Tag :  this.EndValue.Tag);
00584         }
00585     }
00586 
00587     internal class PathActionAnimation :  PrintableActionAnimation<PathAction>
00588     {
00589         public override PathAction StartValue { get; }
00590         public override PathAction EndValue { get; }
00591 
00592         private bool AreTheyTheSame(GraphicsPath pth1, GraphicsPath pth2)
00593         {
00594             if (pth1.Segments.Count != pth2.Segments.Count)
00595             {
00596                 return false;
00597             }
00598 
00599             for (int i = 0; i < pth1.Segments.Count; i++)
00600             {
00601                 if (pth1.Segments[i].Type != pth2.Segments[i].Type)
00602                 {
00603                     return false;
00604                 }
00605             }
00606 
00607             return true;
00608         }
00609 
00610         private Point GetCenter(GraphicsPath path)
00611         {
00612             List<Point> points = path.GetPoints().ToList().Aggregate(new List<Point>(), (a, b) => {
      a.AddRange(b); return a; });
00613 
00614             double cX = 0;
00615             double cY = 0;
00616 
00617             for (int i = 0; i < points.Count; i++)
00618             {
00619                 cX += points[i].X;
00620                 cY += points[i].Y;
00621             }
00622 
00623             return new Point(cX / points.Count, cY / points.Count);
00624         }
00625 
00626         private double ComparePaths(GraphicsPath path1, GraphicsPath path2)
00627         {
00628             Point c1 = GetCenter(path1);
00629             Point c2 = GetCenter(path2);
00630 
00631             return (c1.X - c2.X) * (c1.X - c2.X) + (c1.Y - c2.Y) * (c1.Y - c2.Y);
00632         }
00633 
00634         private List<(int, int)> GetBestAssignment(List<GraphicsPath> figures1, List<GraphicsPath>
      figures2)
00635         {
00636             List<(int, int)> bestAssignment = new List<(int, int)>();
00637 
00638             double[,] distMat = new double[figures1.Count, figures2.Count];
00639 
00640             for (int i = 0; i < figures1.Count; i++)
00641             {
00642                 for (int j = 0; j < figures2.Count; j++)
00643                 {
00644                     distMat[i, j] = ComparePaths(figures1[i], figures2[j]);
00645                 }
00646             }
00647 
00648             List<int> missingIndices1 = Enumerable.Range(0, figures1.Count).ToList();
00649             List<int> missingIndices2 = Enumerable.Range(0, figures2.Count).ToList();
00650 
```

```
00651             if (figures1.Count > figures2.Count)
00652             {
00653                 while (missingIndices2.Count > 0)
00654                 {
00655                     double minValue = double.MaxValue;
00656                     (int, int) bestPair = (-1, -1);
00657 
00658                     for (int i = 0; i < missingIndices1.Count; i++)
00659                     {
00660                         for (int j = 0; j < missingIndices2.Count; j++)
00661                         {
00662                             if (distMat[missingIndices1[i], missingIndices2[j]] < minValue)
00663                             {
00664                                 minValue = distMat[missingIndices1[i], missingIndices2[j]];
00665                                 bestPair = (missingIndices1[i], missingIndices2[j]);
00666                             }
00667                         }
00668                     }
00669 
00670                     bestAssignment.Add(bestPair);
00671                     missingIndices1.Remove(bestPair.Item1);
00672                     missingIndices2.Remove(bestPair.Item2);
00673                 }
00674             }
00675             else
00676             {
00677                 while (missingIndices1.Count > 0)
00678                 {
00679                     double minValue = double.MaxValue;
00680                     (int, int) bestPair = (-1, -1);
00681 
00682                     for (int i = 0; i < missingIndices1.Count; i++)
00683                     {
00684                         for (int j = 0; j < missingIndices2.Count; j++)
00685                         {
00686                             if (distMat[missingIndices1[i], missingIndices2[j]] < minValue)
00687                             {
00688                                 minValue = distMat[missingIndices1[i], missingIndices2[j]];
00689                                 bestPair = (missingIndices1[i], missingIndices2[j]);
00690                             }
00691                         }
00692                     }
00693 
00694                     bestAssignment.Add(bestPair);
00695                     missingIndices1.Remove(bestPair.Item1);
00696                     missingIndices2.Remove(bestPair.Item2);
00697                 }
00698             }
00699 
00700             return bestAssignment;
00701         }
00702 
00703         private void IncreasePoints(List<Point> points, bool isClosed)
00704         {
00705             if (points.Count > 1)
00706             {
00707                 double maxLength = double.MinValue;
00708                 int maxIndex = -1;
00709 
00710                 double target = isClosed ?  points.Count :  points.Count - 1;
00711 
00712                 for (int i = 0; i < target; i++)
00713                 {
00714                     double length = (points[i].X - points[(i + 1) % points.Count].X) * (points[i].X -
      points[(i + 1) % points.Count].X) + (points[i].Y - points[(i + 1) % points.Count].Y) * (points[i].Y -
      points[(i + 1) % points.Count].Y);
00715 
00716                     if (length > maxLength)
00717                     {
00718                         maxLength = length;
00719                         maxIndex = i;
00720                     }
00721                 }
00722 
00723                 points.Insert(maxIndex + 1, new Point((points[maxIndex].X + points[(maxIndex + 1) %
      points.Count].X) * 0.5, (points[maxIndex].Y + points[(maxIndex + 1) % points.Count].Y) * 0.5));
00724             }
00725             else
00726             {
00727                 points.Add(points[0]);
00728             }
00729 
00730         }
00731 
00732         public PathActionAnimation(PathAction startValue, PathAction endValue, double
      linearisationResolution) :  base(startValue, endValue)
00733         {
```

```
00734             GraphicsPath startPath = startValue.Path.ConvertArcsToBeziers();
00735             GraphicsPath endPath = endValue.Path.ConvertArcsToBeziers();
00736 
00737             if (AreTheyTheSame(startPath, endPath))
00738             {
00739                 this.StartValue = new PathAction(startPath, startValue.Fill, startValue.Stroke,
      startValue.LineWidth, startValue.LineCap, startValue.LineJoin, startValue.LineDash, startValue.Tag,
      startValue.FillRule, startValue.IsClipping);
00740                 this.EndValue = new PathAction(endPath, endValue.Fill, endValue.Stroke,
      endValue.LineWidth, endValue.LineCap, endValue.LineJoin, endValue.LineDash, endValue.Tag,
      endValue.FillRule, endValue.IsClipping);
00741             }
00742             else
00743             {
00744                 List<GraphicsPath> startFigures = startPath.GetFigures().ToList();
00745                 List<GraphicsPath> endFigures = endPath.GetFigures().ToList();
00746 
00747                 List<(int, int)> correspondences = GetBestAssignment(startFigures, endFigures);
00748 
00749                 HashSet<int> startFiguresIndices = new HashSet<int>(Enumerable.Range(0,
      startFigures.Count));
00750                 HashSet<int> endFiguresIndices = new HashSet<int>(Enumerable.Range(0,
      endFigures.Count));
00751 
00752                 for (int i = 0; i < correspondences.Count; i++)
00753                 {
00754                     startFiguresIndices.Remove(correspondences[i].Item1);
00755                     endFiguresIndices.Remove(correspondences[i].Item2);
00756                 }
00757 
00758                 foreach (int i in startFiguresIndices)
00759                 {
00760                     Point center = GetCenter(startFigures[i]);
00761 
00762                     GraphicsPath path = new GraphicsPath().MoveTo(center).LineTo(center);
00763                     if (startFigures[i].Segments[startFigures[i].Segments.Count - 1].Type ==
      SegmentType.Close)
00764                     {
00765                         path.Close();
00766                     }
00767 
00768                     endFigures.Add(path);
00769                     correspondences.Add((i, endFigures.Count - 1));
00770                 }
00771 
00772 
00773                 foreach (int i in endFiguresIndices)
00774                 {
00775                     Point center = GetCenter(endFigures[i]);
00776 
00777                     GraphicsPath path = new GraphicsPath().MoveTo(center).LineTo(center);
00778                     if (endFigures[i].Segments[endFigures[i].Segments.Count - 1].Type ==
      SegmentType.Close)
00779                     {
00780                         path.Close();
00781                     }
00782 
00783                     startFigures.Add(path);
00784                     correspondences.Add((startFigures.Count - 1, i));
00785                 }
00786 
00787 
00788                 startPath = new GraphicsPath();
00789                 endPath = new GraphicsPath();
00790 
00791                 for (int i = 0; i < startFigures.Count; i++)
00792                 {
00793                     double startLength = startFigures[correspondences[i].Item1].MeasureLength();
00794                     double endLength = endFigures[correspondences[i].Item2].MeasureLength();
00795 
00796                     double maxLength = Math.Max(startLength, endLength);
00797                     int numPoints = (int)Math.Ceiling(maxLength / linearisationResolution);
00798 
00799                     double startResolution = startLength / maxLength * linearisationResolution;
00800                     double endResolution = endLength / maxLength * linearisationResolution;
00801 
00802                     if (startResolution <= 0)
00803                     {
00804                         startResolution = 1;
00805                     }
00806 
00807                     if (endResolution <= 0)
00808                     {
00809                         endResolution = 1;
00810                     }
00811 
00812                     List<Point> startFigure =
```

```
      startFigures[correspondences[i].Item1].Linearise(startResolution).GetPoints().First();
00813                     bool startIsClosed =
      startFigures[correspondences[i].Item1].Segments[startFigures[correspondences[i].Item1].Segments.Count
      - 1].Type == SegmentType.Close;
00814
00815                     List<Point> endFigure =
      endFigures[correspondences[i].Item2].Linearise(endResolution).GetPoints().First();
00816                     bool endIsClosed =
      endFigures[correspondences[i].Item2].Segments[endFigures[correspondences[i].Item2].Segments.Count -
      1].Type == SegmentType.Close;
00817
00818                     while (startFigure.Count < numPoints)
00819                     {
00820                         IncreasePoints(startFigure, startIsClosed);
00821                     }
00822
00823                     while (endFigure.Count < numPoints)
00824                     {
00825                         IncreasePoints(endFigure, endIsClosed);
00826                     }
00827
00828
00829                     double minVariance = double.MaxValue;
00830                     int bestShift = -1;
00831
00832                     for (int j = 0; j < numPoints; j++)
00833                     {
00834                         double average = 0;
00835                         //double averageSq = 0;
00836
00837                         for (int k = 0; k < numPoints; k++)
00838                         {
00839                             double dist = (startFigure[k].X - endFigure[(k + j) % numPoints].X) *
      (startFigure[k].X - endFigure[(k + j) % numPoints].X) + (startFigure[k].Y - endFigure[(k + j) %
      numPoints].Y) * (startFigure[k].Y - endFigure[(k + j) % numPoints].Y);
00840                             average += dist;
00841                             //averageSq += dist * dist;
00842                         }
00843
00844                         //double variance = averageSq / numPoints - (average / numPoints) * (average /
      numPoints);
00845
00846                         if (average < minVariance)
00847                         {
00848                             minVariance = average;
00849                             bestShift = j;
00850                         }
00851                     }
00852
00853                     startPath.MoveTo(startFigure[0]);
00854                     endPath.MoveTo(endFigure[bestShift % numPoints]);
00855
00856                     for (int j = 0; j < numPoints; j++)
00857                     {
00858                         startPath.LineTo(startFigure[j]);
00859                         endPath.LineTo(endFigure[(j + bestShift) % numPoints]);
00860                     }
00861
00862                     if (startIsClosed && endIsClosed)
00863                     {
00864                         startPath.Close();
00865                         endPath.Close();
00866                     }
00867                     else if (startIsClosed && !endIsClosed)
00868                     {
00869                         startPath.LineTo(startFigure[0]);
00870                         endPath.LineTo(endFigure[endFigure.Count - 1]);
00871                     }
00872                     else if (!startIsClosed && endIsClosed)
00873                     {
00874                         startPath.LineTo(startFigure[startFigure.Count - 1]);
00875                         endPath.LineTo(endFigure[0]);
00876                     }
00877                 }
00878
00879                 this.StartValue = new PathAction(startPath, startValue.Fill, startValue.Stroke,
      startValue.LineWidth, startValue.LineCap, startValue.LineJoin, startValue.LineDash, startValue.Tag,
      startValue.FillRule, startValue.IsClipping);
00880                 this.EndValue = new PathAction(endPath, endValue.Fill, endValue.Stroke,
      endValue.LineWidth, endValue.LineCap, endValue.LineJoin, endValue.LineDash, endValue.Tag,
      endValue.FillRule, endValue.IsClipping);
00881
00882             }
00883         }
00884
00885         private static Segment InterpolateSegment(Segment start, Segment end, double position)
00886         {
```

```
00887             if (start.Type == end.Type)
00888             {
00889                 if (start.Type == SegmentType.Arc)
00890                 {
00891                     ArcSegment startA = start as ArcSegment;
00892                     ArcSegment endA = end as ArcSegment;
00893 
00894                     return new ArcSegment(InterpolationUtils.InterpolatePoint(startA.Points[0],
      endA.Points[0], position), InterpolationUtils.InterpolateDouble(startA.Radius, endA.Radius, position),
      InterpolationUtils.InterpolateDouble(startA.StartAngle, endA.StartAngle, position),
      InterpolationUtils.InterpolateDouble(startA.EndAngle, endA.EndAngle, position));
00895                 }
00896                 else if (start.Type == SegmentType.Move)
00897                 {
00898                     return new MoveSegment(InterpolationUtils.InterpolatePoint(start.Point, end.Point,
      position));
00899                 }
00900                 else if (start.Type == SegmentType.Line)
00901                 {
00902                     return new LineSegment(InterpolationUtils.InterpolatePoint(start.Point, end.Point,
      position));
00903                 }
00904                 else if (start.Type == SegmentType.Close)
00905                 {
00906                     return new CloseSegment();
00907                 }
00908                 else if (start.Type == SegmentType.CubicBezier)
00909                 {
00910                     CubicBezierSegment startC = start as CubicBezierSegment;
00911                     CubicBezierSegment endC = end as CubicBezierSegment;
00912 
00913                     return new
      CubicBezierSegment(InterpolationUtils.InterpolatePoint(startC.Points[0], endC.Points[0], position),
00914                         InterpolationUtils.InterpolatePoint(startC.Points[1], endC.Points[1],
      position),
00915                         InterpolationUtils.InterpolatePoint(startC.Points[2], endC.Points[2],
      position));
00916                 }
00917                 else
00918                 {
00919                     throw new NotImplementedException();
00920                 }
00921             }
00922             else
00923             {
00924                 return position < 0.5 ?  start :  end;
00925             }
00926         }
00927 
00928         protected override PathAction InterpolateConcrete((double, LineJoins, LineCaps, LineDash,
      Brush, Brush) commonElements, double position, Dictionary<string, IEasing> easings)
00929         {
00930             (double lineWidth, LineJoins lineJoin, LineCaps lineCap, LineDash lineDash, Brush stroke,
      Brush fill) = commonElements;
00931 
00932             GraphicsPath path = new GraphicsPath();
00933 
00934             for (int i = 0; i < this.StartValue.Path.Segments.Count; i++)
00935             {
00936                 path.Segments.Add(InterpolateSegment(this.StartValue.Path.Segments[i],
      this.EndValue.Path.Segments[i], position));
00937             }
00938 
00939             return new PathAction(path, fill, stroke, lineWidth, lineCap, lineJoin, lineDash, position
      < 0.5 ?  this.StartValue.Tag :  this.EndValue.Tag, position < 0.5 ?  this.StartValue.FillRule :
      this.EndValue.FillRule, position < 0.5 ?  this.StartValue.IsClipping :  this.EndValue.IsClipping);
00940         }
00941     }
00942 
00943     internal class FilteredGraphicsAnimation :  PrintableActionAnimation<FilteredGraphicsAction>
00944     {
00945         private IGraphicsAnimation GraphicsAnimation { get; }
00946         private IGraphicsAnimation MaskAnimation { get; }
00947 
00948         public FilteredGraphicsAnimation(FilteredGraphicsAction startValue, FilteredGraphicsAction
      endValue, double linearisationResolution) :  base(startValue, endValue)
00949         {
00950             if (startValue.Content == endValue.Content)
00951             {
00952                 this.GraphicsAnimation = new ConstantGraphicsAnimation(startValue.Content);
00953             }
00954             else
00955             {
00956                 this.GraphicsAnimation = new GraphicsAnimation(startValue.Content, endValue.Content,
      linearisationResolution);
00957             }
00958 
```

```
00959             if (startValue.Filter is MaskFilter startM && endValue.Filter is MaskFilter endM)
00960             {
00961                 if (startM.Mask == endM.Mask)
00962                 {
00963                     this.MaskAnimation = new ConstantGraphicsAnimation(startM.Mask);
00964                 }
00965                 else
00966                 {
00967                     this.MaskAnimation = new GraphicsAnimation(startM.Mask, endM.Mask,
      linearisationResolution);
00968                 }
00969             }
00970         }
00971
00972         protected override FilteredGraphicsAction InterpolateConcrete((double, LineJoins, LineCaps,
      LineDash, Brush, Brush) commonElements, double position, Dictionary<string, IEasing> easings)
00973         {
00974             Graphics gpr = this.GraphicsAnimation.Interpolate(position, easings);
00975
00976             if (this.StartValue.Filter.GetType() == this.EndValue.Filter.GetType())
00977             {
00978                 if (this.StartValue.Filter is BoxBlurFilter startBBF && this.EndValue.Filter is
      BoxBlurFilter endBBF)
00979                 {
00980                     BoxBlurFilter filter = new
      BoxBlurFilter(InterpolationUtils.InterpolateDouble(startBBF.BoxRadius, endBBF.BoxRadius, position));
00981
00982                     return new FilteredGraphicsAction(gpr, filter);
00983                 }
00984                 else if (this.StartValue.Filter is ColourMatrixFilter startCMF && this.EndValue.Filter
      is ColourMatrixFilter endCMF)
00985                 {
00986                     double[,] colourMatrix = new double[5, 5];
00987
00988                     for (int i = 0; i < 5; i++)
00989                     {
00990                         for (int j = 0; j < 5; j++)
00991                         {
00992                             colourMatrix[i, j] =
      InterpolationUtils.InterpolateDouble(startCMF.ColourMatrix[i, j], endCMF.ColourMatrix[i, j],
      position);
00993                         }
00994                     }
00995
00996                     ColourMatrixFilter filter = new ColourMatrixFilter(new
      ColourMatrix(colourMatrix));
00997
00998                     return new FilteredGraphicsAction(gpr, filter);
00999                 }
01000                 else if (this.StartValue.Filter is ConvolutionFilter startConvo &&
      this.EndValue.Filter is ConvolutionFilter endConvo)
01001                 {
01002                     if (startConvo.Kernel.GetLength(0) == endConvo.Kernel.GetLength(0) &&
      startConvo.Kernel.GetLength(1) == endConvo.Kernel.GetLength(1))
01003                     {
01004                         double[,] kernel = new double[startConvo.Kernel.GetLength(0),
      startConvo.Kernel.GetLength(1)];
01005
01006                         for (int i = 0; i < kernel.GetLength(0); i++)
01007                         {
01008                             for (int j = 0; j < kernel.GetLength(1); j++)
01009                             {
01010                                 kernel[i, j] =
      InterpolationUtils.InterpolateDouble(startConvo.Kernel[i, j], endConvo.Kernel[i, j], position);
01011                             }
01012                         }
01013
01014                         ConvolutionFilter filter = new ConvolutionFilter(kernel,
      InterpolationUtils.InterpolateDouble(startConvo.Scale, endConvo.Scale, position), position < 0.5 ?
      startConvo.PreserveAlpha :  endConvo.PreserveAlpha,
      InterpolationUtils.InterpolateDouble(startConvo.Normalisation, endConvo.Normalisation, position),
      InterpolationUtils.InterpolateDouble(startConvo.Bias, endConvo.Bias, position));
01015                         return new FilteredGraphicsAction(gpr, filter);
01016                     }
01017                     else
01018                     {
01019                         return position < 0.5 ?  this.StartValue :  this.EndValue;
01020                     }
01021                 }
01022                 else if (this.StartValue.Filter is GaussianBlurFilter startGauss &&
      this.EndValue.Filter is GaussianBlurFilter endGauss)
01023                 {
01024                     GaussianBlurFilter filter = new
      GaussianBlurFilter(InterpolationUtils.InterpolateDouble(startGauss.StandardDeviation,
      endGauss.StandardDeviation, position));
01025                     return new FilteredGraphicsAction(gpr, filter);
01026                 }
```

```
01027                 else if (this.StartValue.Filter is MaskFilter startMask && this.EndValue.Filter is
      MaskFilter endMask)
01028                 {
01029                     Graphics mask = MaskAnimation.Interpolate(position, easings);
01030                     MaskFilter filter = new MaskFilter(mask);
01031                     return new FilteredGraphicsAction(gpr, filter);
01032                 }
01033                 else
01034                 {
01035                     return position < 0.5 ?  this.StartValue :  this.EndValue;
01036                 }
01037             }
01038             else
01039             {
01040                 return position < 0.5 ?  this.StartValue :  this.EndValue;
01041             }
01042         }
01043     }
01044
01045     internal interface IGraphicsAnimation
01046     {
01047         Graphics StartValue { get; }
01048         Graphics EndValue { get; }
01049         Graphics Interpolate(double position, Dictionary<string, IEasing> easings);
01050     }
01051
01052     internal class ConstantGraphicsAnimation :  IGraphicsAnimation
01053     {
01054         public Graphics StartValue { get; }
01055         public Graphics EndValue => StartValue;
01056         public Graphics Interpolate(double position, Dictionary<string, IEasing> easings) =>
      this.StartValue;
01057
01058         public ConstantGraphicsAnimation(Graphics startValue)
01059         {
01060             this.StartValue = startValue;
01061         }
01062     }
01063
01064     internal class GraphicsAnimation :  IGraphicsAnimation
01065     {
01066         public Graphics StartValue { get; }
01067         public Graphics EndValue { get; }
01068
01069         private List<IAnimation> StartAnimations { get; }
01070         private List<IAnimation> EndAnimations { get; }
01071
01072         public GraphicsAnimation(Graphics startValue, Graphics endValue, double
      linearisationResolution)
01073         {
01074             this.StartValue = startValue;
01075             this.EndValue = endValue;
01076
01077             Dictionary<string, IGraphicsAction> startTaggedActions = new Dictionary<string,
      IGraphicsAction>();
01078
01079             foreach (IGraphicsAction action in startValue.Actions)
01080             {
01081                 if (!string.IsNullOrEmpty(action.Tag))
01082                 {
01083                     startTaggedActions[action.Tag] = action;
01084                 }
01085             }
01086
01087             Dictionary<string, IGraphicsAction> endTaggedActions = new Dictionary<string,
      IGraphicsAction>();
01088
01089             foreach (IGraphicsAction action in endValue.Actions)
01090             {
01091                 if (!string.IsNullOrEmpty(action.Tag))
01092                 {
01093                     endTaggedActions[action.Tag] = action;
01094                 }
01095             }
01096
01097             List<IAnimation> startAnimations = new List<IAnimation>();
01098
01099             Dictionary<string, IAnimation> computedAnimations = new Dictionary<string, IAnimation>();
01100
01101             for (int i = 0; i < this.StartValue.Actions.Count; i++)
01102             {
01103                 IGraphicsAction startAction = this.StartValue.Actions[i];
01104
01105                 if (!string.IsNullOrEmpty(this.StartValue.Actions[i].Tag) &&
      endTaggedActions.TryGetValue(this.StartValue.Actions[i].Tag, out IGraphicsAction endAction) &&
      startAction.GetType() == endAction.GetType())
01106                 {
```

```
01107                     if (startAction is TransformAction startT && endAction is TransformAction endT)
01108                     {
01109                         IAnimation animation = new TransformAnimation(startT, endT);
01110 
01111                         computedAnimations[this.StartValue.Actions[i].Tag] = animation;
01112 
01113                         startAnimations.Add(animation);
01114                     }
01115                     else if (startAction is StateAction startS && endAction is StateAction endS)
01116                     {
01117                         IAnimation animation = new StateAnimation(startS, endS);
01118                         computedAnimations[this.StartValue.Actions[i].Tag] = animation;
01119                         startAnimations.Add(animation);
01120                     }
01121                     else if (startAction is TextAction startX && endAction is TextAction endX)
01122                     {
01123                         IAnimation animation = new TextActionAnimation(startX, endX);
01124                         computedAnimations[this.StartValue.Actions[i].Tag] = animation;
01125                         startAnimations.Add(animation);
01126                     }
01127                     else if (startAction is RectangleAction startR && endAction is RectangleAction
      endR)
01128                     {
01129                         IAnimation animation = new RectangleActionAnimation(startR, endR);
01130                         computedAnimations[this.StartValue.Actions[i].Tag] = animation;
01131                         startAnimations.Add(animation);
01132                     }
01133                     else if (startAction is PathAction startP && endAction is PathAction endP)
01134                     {
01135                         IAnimation animation = new PathActionAnimation(startP, endP,
      linearisationResolution);
01136                         computedAnimations[this.StartValue.Actions[i].Tag] = animation;
01137                         startAnimations.Add(animation);
01138                     }
01139                     else if (startAction is RasterImageAction startI && endAction is RasterImageAction
      endI)
01140                     {
01141                         IAnimation animation = new RasterImageActionAnimation(startI, endI);
01142                         computedAnimations[this.StartValue.Actions[i].Tag] = animation;
01143                         startAnimations.Add(animation);
01144                     }
01145                     else if (startAction is FilteredGraphicsAction startF && endAction is
      FilteredGraphicsAction endF)
01146                     {
01147                         IAnimation animation = new FilteredGraphicsAnimation(startF, endF,
      linearisationResolution);
01148                         computedAnimations[this.StartValue.Actions[i].Tag] = animation;
01149                         startAnimations.Add(animation);
01150                     }
01151                     else
01152                     {
01153                         startAnimations.Add(new ConstantAnimation<IGraphicsAction>(startAction));
01154                     }
01155                 }
01156                 else
01157                 {
01158                     startAnimations.Add(new ConstantAnimation<IGraphicsAction>(startAction));
01159                 }
01160             }
01161 
01162 
01163             List<IAnimation> endAnimations = new List<IAnimation>();
01164 
01165             for (int i = 0; i < this.EndValue.Actions.Count; i++)
01166             {
01167                 IGraphicsAction endAction = this.EndValue.Actions[i];
01168 
01169                 if (!string.IsNullOrEmpty(this.EndValue.Actions[i].Tag) &&
      startTaggedActions.TryGetValue(this.EndValue.Actions[i].Tag, out IGraphicsAction startAction) &&
      startAction.GetType() == endAction.GetType())
01170                 {
01171                     endAnimations.Add(computedAnimations[endAction.Tag]);
01172                 }
01173                 else
01174                 {
01175                     endAnimations.Add(new ConstantAnimation<IGraphicsAction>(endAction));
01176                 }
01177             }
01178 
01179             this.StartAnimations = startAnimations;
01180             this.EndAnimations = endAnimations;
01181         }
01182 
01183         public Graphics Interpolate(double position, Dictionary<string, IEasing> easings)
01184         {
01185             if (position <= 0)
01186             {
```

```
01187                 return this.StartValue;
01188             }
01189             else if (position >= 1)
01190             {
01191                 return this.EndValue;
01192             }
01193             else
01194             {
01195                 Graphics gpr = new Graphics();
01196 
01197                 if (position < 0.5)
01198                 {
01199                     for (int i = 0; i < StartAnimations.Count; i++)
01200                     {
01201                         gpr.Actions.Add(StartAnimations[i].Interpolate(position, easings));
01202                     }
01203                 }
01204                 else
01205                 {
01206                     for (int i = 0; i < EndAnimations.Count; i++)
01207                     {
01208                         gpr.Actions.Add(EndAnimations[i].Interpolate(position, easings));
01209                     }
01210                 }
01211 
01212                 return gpr;
01213             }
01214         }
01215     }
01216 
01217 /// <summary>
01218 /// A key frame for an animation.
01219 /// </summary>
01220     public class Frame
01221     {
01222 /// <summary>
01223 /// The duration of the frame, in milliseconds.
01224 /// </summary>
01225         public double Duration { get; }
01226 
01227 /// <summary>
01228 /// The contents of the frame.
01229 /// </summary>
01230         public Graphics Graphics { get; }
01231 
01232 /// <summary>
01233 /// Creates a new <see cref="Frame"/> with the specified contents and duration.
01234 /// </summary>
01235 /// <param name="graphics">The contents of the frame.</param>
01236 /// <param name="duration">The duration of the frame, in milliseconds.</param>
01237         public Frame(Graphics graphics, double duration)
01238         {
01239             this.Graphics = graphics;
01240             this.Duration = duration;
01241         }
01242     }
01243 
01244 /// <summary>
01245 /// Describes a function used to transform the transition speed.
01246 /// </summary>
01247     public interface IEasing
01248     {
01249 /// <summary>
01250 /// Applies the easing to the specified transition offset.
01251 /// </summary>
01252 /// <param name="value">The transition offset (ranging from 0 to 1).</param>
01253 /// <returns>The eased transition offset value.</returns>
01254         [Pure]
01255         double Ease(double value);
01256     }
01257 
01258 /// <summary>
01259 /// Describes an easing defined by a Cubic Bezier curve.
01260 /// </summary>
01261     public class SplineEasing :  IEasing
01262     {
01263 /// <summary>
01264 /// The first control point of the curve.
01265 /// </summary>
01266         public Point ControlPoint1 { get; }
01267 
01268 /// <summary>
01269 /// The second control point of the curve.
01270 /// </summary>
01271         public Point ControlPoint2 { get; }
01272         private double[] SampledEasings { get; }
01273 
```

```
01274 /// <summary>
01275 /// Creates a new <see cref="SplineEasing"/> with the specified control points.  The start point is
      always (0, 0) and the end point is always (1, 1).
01276 /// </summary>
01277 /// <param name="controlPoint1">The first control point of the curve.  Both X and Y must be between 0
      and 1, inclusive.</param>
01278 /// <param name="controlPoint2">The second control point of the curve.  Both X and Y must be between 0
      and 1, inclusive.</param>
01279 /// <exception cref="ArgumentException">This exception is thrown if any coordinate of the control
      points is &lt; 0 or &gt; 1.</exception>
01280         public SplineEasing(Point controlPoint1, Point controlPoint2)
01281         {
01282             if (controlPoint1.X < 0 || controlPoint1.Y < 0 || controlPoint2.X > 1 || controlPoint2.Y >
      1)
01283             {
01284                 throw new ArgumentException("The control point coordinates are out of range!  All
      coordinates must be within 0 and 1 (inclusive).");
01285             }
01286 
01287             this.ControlPoint1 = controlPoint1;
01288             this.ControlPoint2 = controlPoint2;
01289 
01290             Point[] pts = new Point[51];
01291 
01292             for (int i = 0; i <= 50; i++)
01293             {
01294                 double t = i / 50.0;
01295 
01296                 pts[i] = new Point(3 * (1 - t) * (1 - t) * t * controlPoint1.X + 3 * (1 - t) * t * t *
      controlPoint2.X + t * t * t, 3 * (1 - t) * (1 - t) * t * controlPoint1.Y + 3 * (1 - t) * t * t *
      controlPoint2.Y + t * t * t);
01297             }
01298 
01299             double[] sampledEasings = new double[51];
01300 
01301 
01302             double currX = 0;
01303             double currY = pts[0].Y;
01304 
01305             int nextSample = 1;
01306             sampledEasings[0] = pts[0].Y;
01307 
01308             for (int i = 1; i < sampledEasings.Length; i++)
01309             {
01310                 double targetX = nextSample / 50.0;
01311 
01312                 double newX = pts[i].X;
01313 
01314                 while (currX <= targetX && newX >= targetX)
01315                 {
01316                     sampledEasings[nextSample] = currY + (targetX - currX) / (newX - currX) *
      (pts[i].Y - currY);
01317                     nextSample++;
01318                     targetX = nextSample / 50.0;
01319                 }
01320 
01321                 currX = newX;
01322                 currY = pts[i].Y;
01323             }
01324 
01325             this.SampledEasings = sampledEasings;
01326         }
01327 
01328 /// <inheritdoc/>
01329         [Pure]
01330         public double Ease(double value)
01331         {
01332             if (value <= 0)
01333             {
01334                 return 0;
01335             }
01336             else if (value >= 1)
01337             {
01338                 return 1;
01339             }
01340             else
01341             {
01342                 int lowIndex = (int)Math.Floor(value * 50);
01343                 int highIndex = (int)Math.Ceiling(value * 50);
01344 
01345                 return SampledEasings[lowIndex] + (SampledEasings[highIndex] -
      SampledEasings[lowIndex]) * (value * 50 - lowIndex);
01346             }
01347         }
01348     }
01349 
01350 /// <summary>
```

```
01351 /// Describes a linear easing (i.e., no easing).
01352 /// </summary>
01353     public class LinearEasing :  IEasing
01354     {
01355 /// <summary>
01356 /// Creates a new <see cref="LinearEasing"/>.
01357 /// </summary>
01358         public LinearEasing()
01359         {
01360
01361         }
01362
01363 /// <inheritdoc/>
01364         [Pure]
01365         public double Ease(double value) => value;
01366     }
01367
01368 /// <summary>
01369 /// Describes the transition between two successive <see cref="Frame"/>s.
01370 /// </summary>
01371     public class Transition
01372     {
01373 /// <summary>
01374 /// The duration of the transition, in milliseconds.
01375 /// </summary>
01376         public double Duration { get; }
01377
01378 /// <summary>
01379 /// The <see cref="IEasing"/> to apply to all elements for which another easing is not specified.  Set
      to null to use the default linear easing.
01380 /// </summary>
01381         public IEasing OverallEasing { get; } = null;
01382
01383 /// <summary>
01384 /// A dictionary associating graphic action tags to the corresponding <see cref="IEasing"/>.
01385 /// </summary>
01386         public Dictionary<string, IEasing> Easings { get; } = null;
01387
01388 /// <summary>
01389 /// Creates a new <see cref="Transition"/> with the specified duration and easings.
01390 /// </summary>
01391 /// <param name="duration">The duration of the transition, in milliseconds.</param>
01392 /// <param name="easing">The <see cref="IEasing"/> to apply to all elements for which another easing
      is not specified.  Set to null to use the default linear easing.</param>
01393 /// <param name="easings">A dictionary associating graphic action tags to the corresponding <see
      cref="IEasing"/>.</param>
01394         public Transition(double duration, IEasing easing = null, Dictionary<string, IEasing> easings
      = null)
01395         {
01396             if (easing == null)
01397             {
01398                 easing = new LinearEasing();
01399             }
01400
01401             this.Duration = duration;
01402             this.OverallEasing = easing;
01403             this.Easings = easings;
01404         }
01405     }
01406
01407 /// <summary>
01408 /// Describes an animation constituted by a number of frames and transitions between them.
01409 /// </summary>
01410     public class Animation
01411     {
01412 /// <summary>
01413 /// The key frames of the animation.
01414 /// </summary>
01415         public ImmutableList<Frame> Frames { get; private set; } = ImmutableList<Frame>.Empty;
01416
01417 /// <summary>
01418 /// The transitions between successive frames of the animation.  This array always contains one fewer
      element than <see cref="Frames"/>.
01419 /// </summary>
01420         public ImmutableList<Transition> Transitions { get; private set; } =
      ImmutableList<Transition>.Empty;
01421
01422         private ImmutableList<GraphicsAnimation> Animations { get; set; } =
      ImmutableList<GraphicsAnimation>.Empty;
01423
01424
01425 /// <summary>
01426 /// The width of the animation.
01427 /// </summary>
01428         public double Width { get; set; }
01429
01430 /// <summary>
```

```
01431 /// The height of the animation.
01432 /// </summary>
01433         public double Height { get; set; }
01434 
01435 /// <summary>
01436 /// The background colour of the animation.
01437 /// </summary>
01438         public Colour Background { get; set; } = Colour.FromRgba(255, 255, 255, 0);
01439 
01440 /// <summary>
01441 /// The absolute length between successive samples to use when linearising <see
      cref="GraphicsPath"/>s.
01442 /// </summary>
01443         public double LinearisationResolution { get; }
01444 
01445 /// <summary>
01446 /// The total duration of the animation (not including the number of repeats).
01447 /// </summary>
01448         public double Duration { get; private set; } = 0;
01449 
01450 /// <summary>
01451 /// The number of times that the animation should repeat.
01452 /// </summary>
01453         public int RepeatCount { get; set; } = 0;
01454 
01455 /// <summary>
01456 /// Creates a new <see cref="Animation"/> with the specified width, height and linearisation
      resolution.
01457 /// </summary>
01458 /// <param name="width">The width of the animation.</param>
01459 /// <param name="height">The height of the animation.</param>
01460 /// <param name="linearisationResolution">The absolute length between successive samples to use when
      linearising <see cref="GraphicsPath"/>s.</param>
01461         public Animation(double width, double height, double linearisationResolution)
01462         {
01463             this.Width = width;
01464             this.Height = height;
01465             this.LinearisationResolution = linearisationResolution;
01466         }
01467 
01468 /// <summary>
01469 /// Obtains the (interpolated) frame that should be displayed after the specified time has passed
      since the start of the animation.
01470 /// </summary>
01471 /// <param name="time">The time since the start of the animation (in milliseconds).</param>
01472 /// <returns>A <see cref="Page"/> containing the interpolated frame that should be displayed after the
      specified time has passed since the start of the animation.</returns>
01473         public Page GetFrameAtAbsolute(double time)
01474         {
01475             if (time > Duration)
01476             {
01477                 if (this.RepeatCount <= 0)
01478                 {
01479                     time = time % this.Duration;
01480                 }
01481                 else
01482                 {
01483                     time = Math.Max(time % this.Duration, time - this.Duration * RepeatCount);
01484                 }
01485             }
01486 
01487             if (time <= 0)
01488             {
01489                 Page pag = new Page(this.Width, this.Height) { Background = this.Background };
01490                 pag.Graphics.DrawGraphics(0, 0, this.Frames[0].Graphics);
01491                 return pag;
01492             }
01493             else if (time > this.Duration)
01494             {
01495                 Page pag = new Page(this.Width, this.Height) { Background = this.Background };
01496                 pag.Graphics.DrawGraphics(0, 0, this.Frames[this.Frames.Count - 1].Graphics);
01497                 return pag;
01498             }
01499             else
01500             {
01501                 double currTime = 0;
01502 
01503                 for (int i = 0; i < Frames.Count; i++)
01504                 {
01505                     if (i > 0)
01506                     {
01507                         double newTime = currTime + Transitions[i - 1].Duration;
01508 
01509                         if (currTime <= time && newTime >= time)
01510                         {
01511                             Frame previousFrame = Frames[i - 1];
01512 
```

```
01513                           Frame nextFrame = Frames[i];
01514
01515                           if (previousFrame != null && nextFrame != null)
01516                           {
01517                               double position = Transitions[i - 1].OverallEasing.Ease((time -
      currTime) / (newTime - currTime));
01518
01519                               Graphics gpr = Animations[i - 1].Interpolate(position, Transitions[i -
      1].Easings);
01520                               Page pag = new Page(this.Width, this.Height) { Background =
      this.Background };
01521                               pag.Graphics.DrawGraphics(0, 0, gpr);
01522                               return pag;
01523                           }
01524                           else if (previousFrame == null && nextFrame != null)
01525                           {
01526                               Page pag = new Page(this.Width, this.Height) { Background =
      this.Background };
01527                               pag.Graphics.DrawGraphics(0, 0, nextFrame.Graphics);
01528                               return pag;
01529                           }
01530                           else if (previousFrame != null && nextFrame == null)
01531                           {
01532                               Page pag = new Page(this.Width, this.Height) { Background =
      this.Background };
01533                               pag.Graphics.DrawGraphics(0, 0, previousFrame.Graphics);
01534                               return pag;
01535                           }
01536                           else
01537                           {
01538                               Page pag = new Page(this.Width, this.Height) { Background =
      this.Background };
01539                               return pag;
01540                           }
01541                       }
01542
01543                       currTime = newTime;
01544                   }
01545
01546
01547                   {
01548                       double newTime = currTime + Frames[i].Duration;
01549
01550                       if (currTime <= time && newTime >= time)
01551                       {
01552                           Page pag = new Page(this.Width, this.Height) { Background =
      this.Background };
01553                           pag.Graphics.DrawGraphics(0, 0, Frames[i].Graphics);
01554                           return pag;
01555                       }
01556
01557                       currTime = newTime;
01558                   }
01559
01560               }
01561
01562
01563               {
01564                   Page pag = new Page(this.Width, this.Height) { Background = this.Background };
01565                   pag.Graphics.DrawGraphics(0, 0, this.Frames[this.Frames.Count - 1].Graphics);
01566                   return pag;
01567               }
01568           }
01569       }
01570
01571 /// <summary>
01572 /// Obtains the (interpolated) frame that should be displayed after the specified relative time has
      passed since the start of the animation.
01573 /// </summary>
01574 /// <param name="relativeTime">The time since the start of the animation (ranging from 0 for the start
      of the animation, to 1 for the end of the animation).</param>
01575 /// <returns>A <see cref="Page"/> containing the interpolated frame that should be displayed after the
      specified time has passed since the start of the animation.</returns>
01576       public Page GetFrameAtRelative(double relativeTime)
01577       {
01578           return GetFrameAtAbsolute(this.Duration * relativeTime);
01579       }
01580
01581 /// <summary>
01582 /// Adds a new frame to the animation, with the specified transition.
01583 /// </summary>
01584 /// <param name="frame">The new frame to add to the animation.</param>
01585 /// <param name="transition">The transition that should be applied between the previous frame and the
      new frame.  This parameter is ignored for the first frame.  If this is <see langword="null"/>, the
      animation will abruptly change from one frame to the next.</param>
01586       public void AddFrame(Frame frame, Transition transition = null)
01587       {
```

```
01588             this.Frames = this.Frames.Add(frame);
01589             this.Duration += frame.Duration;
01590 
01591             if (this.Frames.Count > 1)
01592             {
01593                 if (transition == null)
01594                 {
01595                     transition = new Transition(0, new LinearEasing());
01596                 }
01597 
01598                 this.Transitions = this.Transitions.Add(transition);
01599                 this.Animations = this.Animations.Add(new
      GraphicsAnimation(this.Frames[this.Frames.Count - 2].Graphics, this.Frames[this.Frames.Count -
      1].Graphics, this.LinearisationResolution));
01600                 this.Duration += transition.Duration;
01601             }
01602         }
01603 
01604 /// <summary>
01605 /// Removes the last frame from the animation (and the corresponding transition).
01606 /// </summary>
01607 /// <exception cref="ArgumentOutOfRangeException">Thrown if there are no frames in the
      animation.</exception>
01608         public void RemoveLastFrame()
01609         {
01610             if (this.Frames.Count > 0)
01611             {
01612                 if (this.Frames.Count > 1)
01613                 {
01614                     double lostDuration = this.Frames[this.Frames.Count - 1].Duration +
      this.Transitions[this.Transitions.Count - 1].Duration;
01615 
01616                     this.Frames = this.Frames.RemoveAt(this.Frames.Count - 1);
01617                     this.Animations = this.Animations.RemoveAt(this.Animations.Count - 1);
01618                     this.Transitions = this.Transitions.RemoveAt(this.Transitions.Count - 1);
01619 
01620                     this.Duration -= lostDuration;
01621                 }
01622                 else
01623                 {
01624                     this.Frames = ImmutableList<Frame>.Empty;
01625                     this.Duration = 0;
01626                 }
01627             }
01628             else
01629             {
01630                 throw new ArgumentOutOfRangeException("There are no frames in the animation!");
01631             }
01632         }
01633     }
01634 }
```

8.68 Brush.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
00019 using System.Collections;
00020 using System.Collections.Generic;
00021 using System.Collections.Immutable;
00022 using System.Diagnostics;
00023 using System.Diagnostics.Contracts;
00024 using System.Linq;
00025 
00026 namespace VectSharp
00027 {
00028 /// <summary>
00029 /// Represents a brush used to fill or stroke graphics elements.  This could be a solid colour, or a
      more complicated gradient or pattern.
```

```
00030 /// </summary>
00031     public abstract class Brush
00032     {
00033         internal Brush() { }
00034 
00035 /// <summary>
00036 /// Returns a brush corresponding the current instance, with the specified <paramref name="opacity"/>
      multiplication applied.
00037 /// </summary>
00038 /// <param name="opacity">The value that will be used to multiply the opacity of the brush.</param>
00039 /// <returns>A brush corresponding the current instance, with the specified <paramref name="opacity"/>
      multiplication applied.</returns>
00040         [Pure]
00041         public abstract Brush MultiplyOpacity(double opacity);
00042 
00043 /// <summary>
00044 /// Implicitly converts a <see cref="Colour"/> into a <see cref="SolidColourBrush"/>.
00045 /// </summary>
00046 /// <param name="colour">The <see cref="Colour"/> to use for the brush.</param>
00047         public static implicit operator Brush(Colour colour)
00048         {
00049             return new SolidColourBrush(colour);
00050         }
00051     }
00052 
00053 /// <summary>
00054 /// Represents a brush painting with a single solid colour.
00055 /// </summary>
00056     [DebuggerDisplay("{Colour}")]
00057     public class SolidColourBrush :  Brush
00058     {
00059 /// <summary>
00060 /// The colour of the brush.
00061 /// </summary>
00062         public Colour Colour { get; }
00063 
00064 /// <summary>
00065 /// Red component of the colour.  Range:  [0, 1].
00066 /// </summary>
00067         public double R => Colour.R;
00068 
00069 /// <summary>
00070 /// Green component of the colour.  Range:  [0, 1].
00071 /// </summary>
00072         public double G => Colour.G;
00073 
00074 /// <summary>
00075 /// Blue component of the colour.  Range:  [0, 1].
00076 /// </summary>
00077         public double B => Colour.B;
00078 
00079 /// <summary>
00080 /// Alpha component of the colour.  Range:  [0, 1].
00081 /// </summary>
00082         public double A => Colour.A;
00083 
00084 /// <summary>
00085 /// Creates a new <see cref="SolidColourBrush"/> with the specified <paramref name="colour"/>.
00086 /// </summary>
00087 /// <param name="colour">The <see cref="Colour"/> to use for the brush.</param>
00088         public SolidColourBrush(Colour colour)
00089         {
00090             this.Colour = colour;
00091         }
00092 
00093 /// <inheritdoc/>
00094         public override Brush MultiplyOpacity(double opacity)
00095         {
00096             return new SolidColourBrush(this.Colour.WithAlpha(this.Colour.A * opacity));
00097         }
00098 
00099 /// <summary>
00100 /// Implicitly converts a <see cref="Colour"/> into a <see cref="SolidColourBrush"/>.
00101 /// </summary>
00102 /// <param name="colour">The <see cref="Colour"/> to use for the brush.</param>
00103         public static implicit operator SolidColourBrush(Colour colour)
00104         {
00105             return new SolidColourBrush(colour);
00106         }
00107     }
00108 
00109 /// <summary>
00110 /// Represents a colour stop in a gradient.
00111 /// </summary>
00112     [DebuggerDisplay("\\{ Colour:  {Colour}, Offset:  {Offset} \\}")]
00113     public struct GradientStop
00114     {
```

```
00115 /// <summary>
00116 /// The <see cref="Colour"/> at the gradient stop.
00117 /// </summary>
00118         public Colour Colour { get; }
00119 
00120 /// <summary>
00121 /// The offset of the gradient stop.  Range:  [0, 1].
00122 /// </summary>
00123         public double Offset { get; }
00124 
00125 /// <summary>
00126 /// Creates a new <see cref="GradientStop"/> instance.
00127 /// </summary>
00128 /// <param name="colour">The <see cref="Colour"/> at the gradient stop.</param>
00129 /// <param name="offset">The offset of the gradient stop.  Range:  [0, 1].</param>
00130         public GradientStop(Colour colour, double offset)
00131         {
00132             this.Colour = colour;
00133             this.Offset = Math.Max(0, Math.Min(1, offset));
00134         }
00135 
00136 /// <summary>
00137 /// Returns a <see cref="GradientStop"/> corresponding to the current instance, whose colour's opacity
      has been multiplied by the specified value.
00138 /// </summary>
00139 /// <param name="opacity">The value that will be used to multiply the colour's opacity.</param>
00140 /// <returns>A <see cref="GradientStop"/> corresponding to the current instance, whose colour's
      opacity has been multiplied by the specified value.</returns>
00141         [Pure]
00142         public GradientStop MultiplyOpacity(double opacity)
00143         {
00144             return new GradientStop(this.Colour.WithAlpha(this.Colour.A * opacity), this.Offset);
00145         }
00146     }
00147 
00148 /// <summary>
00149 /// Represents a read-only list of <see cref="GradientStop"/>s.
00150 /// </summary>
00151     [DebuggerDisplay("\\{ {Count} stops \\}")]
00152     public class GradientStops :  IReadOnlyList<GradientStop>
00153     {
00154 /// <summary>
00155 /// The minimum distance that is enforced between consecutive gradient stops.
00156 /// </summary>
00157         public static readonly double StopTolerance = 1e-7;
00158 
00159 /// <inheritdoc/>
00160         public GradientStop this[int index] => gradientStops[index];
00161 
00162 /// <inheritdoc/>
00163         public int Count => gradientStops.Count;
00164 
00165         private ImmutableList<GradientStop> gradientStops { get; set; }
00166 
00167 /// <inheritdoc/>
00168         public IEnumerator<GradientStop> GetEnumerator()
00169         {
00170             return ((IEnumerable<GradientStop>)gradientStops).GetEnumerator();
00171         }
00172 
00173         IEnumerator IEnumerable.GetEnumerator()
00174         {
00175             return ((IEnumerable)gradientStops).GetEnumerator();
00176         }
00177 
00178 /// <summary>
00179 /// Creates a new <see cref="GradientStops"/> instance containing the specified gradient stops.
00180 /// </summary>
00181 /// <param name="gradientStops">The gradient stops that will be contained in the <see
      cref="GradientStops"/> object.</param>
00182         public GradientStops(IEnumerable<GradientStop> gradientStops)
00183         {
00184             List<GradientStop> stops = (from el in gradientStops orderby el.Offset ascending select
      el).ToList();
00185 
00186             if (stops.Count == 0)
00187             {
00188                 stops.Add(new GradientStop(Colour.FromRgba(0, 0, 0, 0), 0));
00189             }
00190 
00191             if (stops[0].Offset > 0)
00192             {
00193                 stops.Insert(0, new GradientStop(stops[0].Colour, 0));
00194             }
00195 
00196             if (stops[stops.Count - 1].Offset < 1)
00197             {
```

```
00198                 stops.Add(new GradientStop(stops[stops.Count - 1].Colour, 1));
00199             }
00200 
00201             for (int i = 1; i < stops.Count - 1; i++)
00202             {
00203                 bool closeToPrevious = (stops[i].Offset - stops[i - 1].Offset < StopTolerance);
00204                 bool closeToNext = (stops[i + 1].Offset - stops[i].Offset < StopTolerance);
00205 
00206                 if (closeToPrevious && !closeToNext)
00207                 {
00208                     stops[i] = new GradientStop(stops[i].Colour, stops[i - 1].Offset + StopTolerance);
00209                 }
00210                 else if (!closeToPrevious && closeToNext)
00211                 {
00212                     stops[i] = new GradientStop(stops[i].Colour, stops[i + 1].Offset - StopTolerance);
00213                 }
00214                 else if (closeToPrevious && closeToNext)
00215                 {
00216                     stops.RemoveAt(i);
00217                     i--;
00218                 }
00219             }
00220 
00221             this.gradientStops = ImmutableList.Create(stops.ToArray());
00222         }
00223 
00224 /// <summary>
00225 /// Creates a new <see cref="GradientStops"/> instance containing the specified gradient stops.
00226 /// </summary>
00227 /// <param name="gradientStops">The gradient stops that will be contained in the <see
      cref="GradientStops"/> object.</param>
00228         public GradientStops(params GradientStop[] gradientStops) :
      this((IEnumerable<GradientStop>)gradientStops)
00229         {
00230 
00231         }
00232 
00233 /// <summary>
00234 /// Gets the colour at a certain position on the gradient.
00235 /// </summary>
00236 /// <param name="position">The position in the gradient (ranging from 0 to 1).</param>
00237 /// <returns>The colour of the gradient at the specified position.</returns>
00238         public Colour GetColourAt(double position)
00239         {
00240             if (position <= 0)
00241             {
00242                 return this.gradientStops[0].Colour;
00243             }
00244             else if (position >= 1)
00245             {
00246                 return this.gradientStops[this.gradientStops.Count - 1].Colour;
00247             }
00248             else
00249             {
00250                 for (int i = 1; i < this.gradientStops.Count; i++)
00251                 {
00252                     if (this.gradientStops[i - 1].Offset < position && this.gradientStops[i].Offset >=
      position)
00253                     {
00254                         double factor = (position - this.gradientStops[i - 1].Offset) /
      (this.gradientStops[i].Offset - this.gradientStops[i - 1].Offset);
00255 
00256                         return Colour.FromRgba(this.gradientStops[i - 1].Colour.R * (1 - factor) +
      this.gradientStops[i].Colour.R * factor,
00257                             this.gradientStops[i - 1].Colour.G * (1 - factor) +
      this.gradientStops[i].Colour.G * factor,
00258                             this.gradientStops[i - 1].Colour.B * (1 - factor) +
      this.gradientStops[i].Colour.B * factor,
00259                             this.gradientStops[i - 1].Colour.A * (1 - factor) +
      this.gradientStops[i].Colour.A * factor);
00260                     }
00261                 }
00262                 return Colour.FromRgba(0, 0, 0, 0);
00263             }
00264         }
00265 
00266 /// <summary>
00267 /// Convert a <see cref="GradientStops"/> object to a function that assigns a colour to values between
      0 and 1.
00268 /// </summary>
00269 /// <param name="stops">The <see cref="GradientStops"/> to convert.</param>
00270         public static implicit operator Func<double, Colour>(GradientStops stops) =>
      stops.GetColourAt;
00271     }
00272 
00273 /// <summary>
00274 /// Represents a brush painting with a gradient.
```

```
00275 /// </summary>
00276     [DebuggerDisplay("\\{ GradientStops:  {GradientStops} \\}")]
00277     public abstract class GradientBrush :  Brush
00278     {
00279 /// <summary>
00280 /// The colour stops in the gradient.
00281 /// </summary>
00282         public GradientStops GradientStops { get; protected internal set; }
00283 
00284         internal GradientBrush() { }
00285     }
00286 
00287 /// <summary>
00288 /// Represents a brush painting with a linear gradient.
00289 /// </summary>
00290     [DebuggerDisplay("\\{ StartPoint:  {StartPoint}, EndPoint:  {EndPoint}, GradientStops:
      {GradientStops} \\}")]
00291     public class LinearGradientBrush :  GradientBrush
00292     {
00293 /// <summary>
00294 /// The starting point of the gradient.  Note that this is relative to the current coordinate system
      when the gradient is used.
00295 /// </summary>
00296         public Point StartPoint { get; }
00297 
00298 /// <summary>
00299 /// The end point of the gradient.  Note that this is relative to the current coordinate system when
      the gradient is used.
00300 /// </summary>
00301         public Point EndPoint { get; }
00302 
00303 /// <summary>
00304 /// Creates a new <see cref="LinearGradientBrush"/> with the specified start point, end point and
      gradient stops.
00305 /// </summary>
00306 /// <param name="startPoint">The starting point of the gradient.  Note that this is relative to the
      current coordinate system when the gradient is used.</param>
00307 /// <param name="endPoint">The ending point of the gradient.  Note that this is relative to the
      current coordinate system when the gradient is used.</param>
00308 /// <param name="gradientStops">The colour stops in the gradient.</param>
00309         public LinearGradientBrush(Point startPoint, Point endPoint, IEnumerable<GradientStop>
      gradientStops)
00310         {
00311             this.StartPoint = startPoint;
00312             this.EndPoint = endPoint;
00313 
00314             this.GradientStops = new GradientStops(gradientStops);
00315         }
00316 
00317 /// <summary>
00318 /// Creates a new <see cref="LinearGradientBrush"/> with the specified start point, end point and
      gradient stops.
00319 /// </summary>
00320 /// <param name="startPoint">The starting point of the gradient.  Note that this is relative to the
      current coordinate system when the gradient is used.</param>
00321 /// <param name="endPoint">The ending point of the gradient.  Note that this is relative to the
      current coordinate system when the gradient is used.</param>
00322 /// <param name="gradientStops">The colour stops in the gradient.</param>
00323         public LinearGradientBrush(Point startPoint, Point endPoint, params GradientStop[]
      gradientStops)
00324         {
00325             this.StartPoint = startPoint;
00326             this.EndPoint = endPoint;
00327             List<GradientStop> stops = (from el in gradientStops orderby el.Offset ascending select
      el).ToList();
00328 
00329             if (stops.Count == 0)
00330             {
00331                 stops.Add(new GradientStop(Colour.FromRgba(0, 0, 0, 0), 0));
00332             }
00333 
00334             if (stops[0].Offset > 0)
00335             {
00336                 stops.Insert(0, new GradientStop(stops[0].Colour, 0));
00337             }
00338 
00339             if (stops[stops.Count - 1].Offset < 1)
00340             {
00341                 stops.Add(new GradientStop(stops[stops.Count - 1].Colour, 1));
00342             }
00343 
00344             this.GradientStops = new GradientStops(gradientStops);
00345         }
00346 
00347 /// <summary>
00348 /// Returns a <see cref="LinearGradientBrush"/> with the same gradient stops as the current instance,
      whose start and end point correspond to the points of the current instance in the
```

```
00349 /// original reference frame of the <paramref name="referenceGraphics"/>.  This involves computing the current transform matrix of the <paramref name="referenceGraphics" />, inverting it,
00350 /// and applying the inverse matrix to the <see cref="StartPoint"/> and <see cref="EndPoint"/> of the current instance.
00351 /// </summary>
00352 /// <param name="referenceGraphics">The <see cref="Graphics"/> whose original reference frame is to be used.</param>
00353 /// <returns>A <see cref="LinearGradientBrush"/> with the same gradient stops as the current instance, whose start and end point correspond to the points of the current instance in the
00354 /// original reference frame of the <paramref name="referenceGraphics"/>.</returns>
00355         [Pure]
00356         public LinearGradientBrush RelativeTo(Graphics referenceGraphics)
00357         {
00358             Stack<double[,]> transformMatrix = new Stack<double[,]>();
00359             double[,] currMatrix = new double[3, 3] { { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0, 1 } };
00360 
00361             for (int i = 0; i < referenceGraphics.Actions.Count; i++)
00362             {
00363                 if (referenceGraphics.Actions[i] is TransformAction)
00364                 {
00365                     TransformAction trf = referenceGraphics.Actions[i] as TransformAction;
00366 
00367                     if (trf.Delta != null)
00368                     {
00369                         currMatrix = Graphics.Multiply(currMatrix, Graphics.TranslationMatrix(trf.Delta.Value.X, trf.Delta.Value.Y));
00370                     }
00371                     else if (trf.Angle != null)
00372                     {
00373                         currMatrix = Graphics.Multiply(currMatrix, Graphics.RotationMatrix(trf.Angle.Value));
00374                     }
00375                     else if (trf.Scale != null)
00376                     {
00377                         currMatrix = Graphics.Multiply(currMatrix, Graphics.ScaleMatrix(trf.Scale.Value.Width, trf.Scale.Value.Height));
00378                     }
00379                     else if (trf.Matrix != null)
00380                     {
00381                         currMatrix = Graphics.Multiply(currMatrix, trf.Matrix);
00382                     }
00383                 }
00384                 else if (referenceGraphics.Actions[i] is StateAction)
00385                 {
00386                     if (((StateAction)referenceGraphics.Actions[i]).StateActionType == StateAction.StateActionTypes.Save)
00387                     {
00388                         transformMatrix.Push(currMatrix);
00389                     }
00390                     else
00391                     {
00392                         currMatrix = transformMatrix.Pop();
00393                     }
00394                 }
00395             }
00396 
00397             currMatrix = Graphics.Invert(currMatrix);
00398 
00399             Point p1 = Graphics.Multiply(currMatrix, this.StartPoint);
00400             Point p2 = Graphics.Multiply(currMatrix, this.EndPoint);
00401 
00402             return new LinearGradientBrush(p1, p2, this.GradientStops);
00403         }
00404 
00405 /// <inheritdoc/>
00406         public override Brush MultiplyOpacity(double opacity)
00407         {
00408             return new LinearGradientBrush(this.StartPoint, this.EndPoint, from el in this.GradientStops select el.MultiplyOpacity(opacity));
00409         }
00410     }
00411 
00412 /// <summary>
00413 /// Represents a brush painting with a radial gradient.
00414 /// </summary>
00415     [DebuggerDisplay("\\{ FocalPoint:  {FocalPoint}, Centre:  {Centre}, Radius:  {Radius}, GradientStops:  {GradientStops} \\}")]
00416     public class RadialGradientBrush :  GradientBrush
00417     {
00418 /// <summary>
00419 /// The focal point of the gradient (i.e.  the point within the circle where the gradient starts).
00420 /// </summary>
00421         public Point FocalPoint { get; }
00422 
00423 /// <summary>
00424 /// Represents the centre of the gradient.
00425 /// </summary>
```

```
00426         public Point Centre { get; }
00427
00428 /// <summary>
00429 /// The radius of the gradient.
00430 /// </summary>
00431         public double Radius { get; }
00432
00433 /// <summary>
00434 /// Creates a new <see cref="RadialGradientBrush"/> with the specified focal point, centre, radius and
      gradient stops.
00435 /// </summary>
00436 /// <param name="focalPoint">The focal point of the gradient.  Note that this is relative to the
      current coordinate system when the gradient is used.</param>
00437 /// <param name="centre">The centre of the gradient.  Note that this is relative to the current
      coordinate system when the gradient is used.</param>
00438 /// <param name="radius">The radius of the gradient.  Note that this is relative to the current
      coordinate system when the gradient is used.</param>
00439 /// <param name="gradientStops">The colour stops in the gradient.</param>
00440         public RadialGradientBrush(Point focalPoint, Point centre, double radius, params
      GradientStop[] gradientStops)
00441         {
00442             if (new Point(focalPoint.X - centre.X, focalPoint.Y - centre.Y).Modulus() > radius)
00443             {
00444                 Point norm = new Point(focalPoint.X - centre.X, focalPoint.Y - centre.Y).Normalize();
00445                 focalPoint = new Point(centre.X + norm.X * radius, centre.Y + norm.Y * radius);
00446             }
00447
00448             this.FocalPoint = focalPoint;
00449             this.Centre = centre;
00450             this.Radius = radius;
00451
00452             List<GradientStop> stops = (from el in gradientStops orderby el.Offset ascending select
      el).ToList();
00453
00454             if (stops.Count == 0)
00455             {
00456                 stops.Add(new GradientStop(Colour.FromRgba(0, 0, 0, 0), 0));
00457             }
00458
00459             if (stops[0].Offset > 0)
00460             {
00461                 stops.Insert(0, new GradientStop(stops[0].Colour, 0));
00462             }
00463
00464             if (stops[stops.Count - 1].Offset < 1)
00465             {
00466                 stops.Add(new GradientStop(stops[stops.Count - 1].Colour, 1));
00467             }
00468
00469             this.GradientStops = new GradientStops(gradientStops);
00470         }
00471
00472 /// <summary>
00473 /// Creates a new <see cref="RadialGradientBrush"/> with the specified focal point, centre, radius and
      gradient stops.
00474 /// </summary>
00475 /// <param name="focalPoint">The focal point of the gradient.  Note that this is relative to the
      current coordinate system when the gradient is used.</param>
00476 /// <param name="centre">The centre of the gradient.  Note that this is relative to the current
      coordinate system when the gradient is used.</param>
00477 /// <param name="radius">The radius of the gradient.  Note that this is relative to the current
      coordinate system when the gradient is used.</param>
00478 /// <param name="gradientStops">The colour stops in the gradient.</param>
00479         public RadialGradientBrush(Point focalPoint, Point centre, double radius,
      IEnumerable<GradientStop> gradientStops)
00480         {
00481             if (new Point(focalPoint.X - centre.X, focalPoint.Y - centre.Y).Modulus() > radius)
00482             {
00483                 Point norm = new Point(focalPoint.X - centre.X, focalPoint.Y - centre.Y).Normalize();
00484                 focalPoint = new Point(centre.X + norm.X * radius, centre.Y + norm.Y * radius);
00485             }
00486
00487             this.FocalPoint = focalPoint;
00488             this.Centre = centre;
00489             this.Radius = radius;
00490
00491             List<GradientStop> stops = (from el in gradientStops orderby el.Offset ascending select
      el).ToList();
00492
00493             if (stops.Count == 0)
00494             {
00495                 stops.Add(new GradientStop(Colour.FromRgba(0, 0, 0, 0), 0));
00496             }
00497
00498             if (stops[0].Offset > 0)
00499             {
00500                 stops.Insert(0, new GradientStop(stops[0].Colour, 0));
```

```
00501             }
00502
00503             if (stops[stops.Count - 1].Offset < 1)
00504             {
00505                 stops.Add(new GradientStop(stops[stops.Count - 1].Colour, 1));
00506             }
00507
00508             this.GradientStops = new GradientStops(gradientStops);
00509         }
00510
00511 /// <inheritdoc/>
00512         public override Brush MultiplyOpacity(double opacity)
00513         {
00514             return new RadialGradientBrush(this.FocalPoint, this.Centre, this.Radius, from el in
      this.GradientStops select el.MultiplyOpacity(opacity));
00515         }
00516     }
00517 }
```

8.69 Colour.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Diagnostics;
00020 using System.Diagnostics.Contracts;
00021
00022 namespace VectSharp
00023 {
00024 /// <summary>
00025 /// Represents an RGB colour.
00026 /// </summary>
00027     [DebuggerDisplay("{DebuggerDisplay,nq}")]
00028     public partial struct Colour :  IEquatable<Colour>
00029     {
00030 /// <summary>
00031 /// Red component of the colour.  Range:  [0, 1].
00032 /// </summary>
00033         public double R;
00034
00035 /// <summary>
00036 /// Green component of the colour.  Range:  [0, 1].
00037 /// </summary>
00038         public double G;
00039
00040 /// <summary>
00041 /// Blue component of the colour.  Range:  [0, 1].
00042 /// </summary>
00043         public double B;
00044
00045 /// <summary>
00046 /// Alpha component of the colour.  Range:  [0, 1].
00047 /// </summary>
00048         public double A;
00049
00050         private Colour(double r, double g, double b, double a)
00051         {
00052             R = r;
00053             G = g;
00054             B = b;
00055             A = a;
00056         }
00057
00058 /// <summary>
00059 /// Create a new colour from RGB (red, green and blue) values.
00060 /// </summary>
00061 /// <param name="r">The red component of the colour.  Range:  [0, 1].</param>
00062 /// <param name="g">The green component of the colour.  Range:  [0, 1].</param>
```

```
00063 /// <param name="b">The blue component of the colour.  Range:  [0, 1].</param>
00064 /// <returns>A <see cref="Colour"/> struct with the specified components and an alpha component of
      1.</returns>
00065         public static Colour FromRgb(double r, double g, double b)
00066         {
00067             return new Colour(r, g, b, 1);
00068         }
00069 
00070 /// <summary>
00071 /// Create a new colour from RGB (red, green and blue) values.
00072 /// </summary>
00073 /// <param name="r">The red component of the colour.  Range:  [0, 255].</param>
00074 /// <param name="g">The green component of the colour.  Range:  [0, 255].</param>
00075 /// <param name="b">The blue component of the colour.  Range:  [0, 255].</param>
00076 /// <returns>A <see cref="Colour"/> struct with the specified components and an alpha component of
      1.</returns>
00077         public static Colour FromRgb(byte r, byte g, byte b)
00078         {
00079             return new Colour(r / 255.0, g / 255.0, b / 255.0, 1);
00080         }
00081 
00082 /// <summary>
00083 /// Create a new colour from RGB (red, green and blue) values.
00084 /// </summary>
00085 /// <param name="r">The red component of the colour.  Range:  [0, 255].</param>
00086 /// <param name="g">The green component of the colour.  Range:  [0, 255].</param>
00087 /// <param name="b">The blue component of the colour.  Range:  [0, 255].</param>
00088 /// <returns>A <see cref="Colour"/> struct with the specified components and an alpha component of
      1.</returns>
00089         public static Colour FromRgb(int r, int g, int b)
00090         {
00091             return new Colour(r / 255.0, g / 255.0, b / 255.0, 1);
00092         }
00093 
00094 /// <summary>
00095 /// Create a new colour from RGBA (red, green, blue and alpha) values.
00096 /// </summary>
00097 /// <param name="r">The red component of the colour.  Range:  [0, 1].</param>
00098 /// <param name="g">The green component of the colour.  Range:  [0, 1].</param>
00099 /// <param name="b">The blue component of the colour.  Range:  [0, 1].</param>
00100 /// <param name="a">The alpha component of the colour.  Range:  [0, 1].</param>
00101 /// <returns>A <see cref="Colour"/> struct with the specified components.</returns>
00102         public static Colour FromRgba(double r, double g, double b, double a)
00103         {
00104             return new Colour(r, g, b, a);
00105         }
00106 
00107 /// <summary>
00108 /// Create a new colour from RGBA (red, green, blue and alpha) values.
00109 /// </summary>
00110 /// <param name="r">The red component of the colour.  Range:  [0, 255].</param>
00111 /// <param name="g">The green component of the colour.  Range:  [0, 255].</param>
00112 /// <param name="b">The blue component of the colour.  Range:  [0, 255].</param>
00113 /// <param name="a">The alpha component of the colour.  Range:  [0, 255].</param>
00114 /// <returns>A <see cref="Colour"/><see cref="Colour"/> struct with the specified
      components.</returns>
00115         public static Colour FromRgba(byte r, byte g, byte b, byte a)
00116         {
00117             return new Colour(r / 255.0, g / 255.0, b / 255.0, a / 255.0);
00118         }
00119 
00120 /// <summary>
00121 /// Create a new colour from RGBA (red, green, blue and alpha) values.
00122 /// </summary>
00123 /// <param name="r">The red component of the colour.  Range:  [0, 255].</param>
00124 /// <param name="g">The green component of the colour.  Range:  [0, 255].</param>
00125 /// <param name="b">The blue component of the colour.  Range:  [0, 255].</param>
00126 /// <param name="a">The alpha component of the colour.  Range:  [0, 1].</param>
00127 /// <returns>A <see cref="Colour"/> struct with the specified components.</returns>
00128         public static Colour FromRgba(byte r, byte g, byte b, double a)
00129         {
00130             return new Colour(r / 255.0, g / 255.0, b / 255.0, a);
00131         }
00132 /// <summary>
00133 /// Create a new colour from RGBA (red, green, blue and alpha) values.
00134 /// </summary>
00135 /// <param name="r">The red component of the colour.  Range:  [0, 255].</param>
00136 /// <param name="g">The green component of the colour.  Range:  [0, 255].</param>
00137 /// <param name="b">The blue component of the colour.  Range:  [0, 255].</param>
00138 /// <param name="a">The alpha component of the colour.  Range:  [0, 255].</param>
00139 /// <returns>A <see cref="Colour"/> struct with the specified components.</returns>
00140         public static Colour FromRgba(int r, int g, int b, int a)
00141         {
00142             return new Colour(r / 255.0, g / 255.0, b / 255.0, a / 255.0);
00143         }
00144 
00145 /// <summary>
```

```
00146 /// Create a new colour from RGBA (red, green, blue and alpha) values.
00147 /// </summary>
00148 /// <param name="r">The red component of the colour.  Range:  [0, 255].</param>
00149 /// <param name="g">The green component of the colour.  Range:  [0, 255].</param>
00150 /// <param name="b">The blue component of the colour.  Range:  [0, 255].</param>
00151 /// <param name="a">The alpha component of the colour.  Range:  [0, 1].</param>
00152 /// <returns>A <see cref="Colour"/> struct with the specified components.</returns>
00153         public static Colour FromRgba(int r, int g, int b, double a)
00154         {
00155             return new Colour(r / 255.0, g / 255.0, b / 255.0, a);
00156         }
00157 
00158 /// <summary>
00159 /// Create a new colour from RGBA (red, green, blue and alpha) values.
00160 /// </summary>
00161 /// <param name="colour">A <see cref="ValueTuple{Int32, Int32, Int32, Double}"/> containing component
      information for the colour.  For r, g, and b, range:  [0, 255]; for a, range:  [0, 1].</param>
00162 /// <returns>A <see cref="Colour"/> struct with the specified components.</returns>
00163         public static Colour FromRgba((int r, int g, int b, double a) colour)
00164         {
00165             return new Colour(colour.r / 255.0, colour.g / 255.0, colour.b / 255.0, colour.a);
00166         }
00167 
00168 /// <inheritdoc/>
00169         public override bool Equals(object obj)
00170         {
00171             if (!(obj is Colour))
00172             {
00173                 return false;
00174             }
00175             else
00176             {
00177                 return this.Equals((Colour)obj);
00178             }
00179         }
00180 
00181 /// <inheritdoc/>
00182         public bool Equals(Colour col)
00183         {
00184             return col.R == this.R && col.G == this.G && col.B == this.B && col.A == this.A;
00185         }
00186 
00187 /// <inheritdoc/>
00188         public static bool operator ==(Colour col1, Colour col2)
00189         {
00190             return col1.R == col2.R && col1.G == col2.G && col1.B == col2.B && col1.A == col2.A;
00191         }
00192 
00193 /// <inheritdoc/>
00194         public static bool operator !=(Colour col1, Colour col2)
00195         {
00196             return col1.R != col2.R || col1.G != col2.G || col1.B != col2.B || col1.A != col2.A;
00197         }
00198 
00199 /// <inheritdoc/>
00200         public override int GetHashCode()
00201         {
00202             return (int)(this.R * 255 + this.G * 255 * 255 + this.B * 255 * 255 * 255 + this.A * 255 *
      255 * 255 * 255);
00203         }
00204 
00205 /// <summary>
00206 /// Convert the <see cref="Colour"/> object into a hex string that is constituted by a "#" followed by
      two-digit hexadecimal representations of the red, green and blue components of the colour (in the
      range 0x00 - 0xFF).
00207 /// Optionally also includes opacity (alpha channel) data.
00208 /// </summary>
00209 /// <param name="includeAlpha">Whether two additional hex digits representing the colour's opacity
      (alpha channel) should be included in the string.</param>
00210 /// <returns>A hex colour string.</returns>
00211         public string ToCSSString(bool includeAlpha)
00212         {
00213             if (includeAlpha)
00214             {
00215                 return "#" + ((int)Math.Round(this.R * 255)).ToString("X2") + ((int)Math.Round(this.G
      * 255)).ToString("X2") + ((int)Math.Round(this.B * 255)).ToString("X2") + ((int)Math.Round(this.A *
      255)).ToString("X2");
00216             }
00217             else
00218             {
00219                 return "#" + ((int)Math.Round(this.R * 255)).ToString("X2") + ((int)Math.Round(this.G
      * 255)).ToString("X2") + ((int)Math.Round(this.B * 255)).ToString("X2");
00220             }
00221         }
00222 
00223 /// <summary>
00224 /// Convert a CSS colour string into a <see cref="Colour"/> object.
```

```
00225 /// </summary>
00226 /// <param name="cssString">The CSS colour string.  In addition to 148 standard colour names
      (case-insensitive), #RGB, #RGBA, #RRGGBB and #RRGGBBAA hex strings and rgb(r, g, b) and rgba(r, g, b,
      a) functional colour notations are supported.</param>
00227 /// <returns></returns>
00228         public static Colour?  FromCSSString(string cssString)
00229         {
00230             if (cssString.StartsWith("#"))
00231             {
00232                 cssString = cssString.Substring(1);
00233 
00234                 try
00235                 {
00236                     if (cssString.Length == 3)
00237                     {
00238                         byte r = byte.Parse(cssString.Substring(0, 1) + cssString.Substring(0, 1),
      System.Globalization.NumberStyles.HexNumber);
00239                         byte g = byte.Parse(cssString.Substring(1, 1) + cssString.Substring(1, 1),
      System.Globalization.NumberStyles.HexNumber);
00240                         byte b = byte.Parse(cssString.Substring(2, 1) + cssString.Substring(2, 1),
      System.Globalization.NumberStyles.HexNumber);
00241 
00242                         return Colour.FromRgb(r, g, b);
00243                     }
00244                     else if (cssString.Length == 4)
00245                     {
00246                         byte r = byte.Parse(cssString.Substring(0, 1) + cssString.Substring(0, 1),
      System.Globalization.NumberStyles.HexNumber);
00247                         byte g = byte.Parse(cssString.Substring(1, 1) + cssString.Substring(1, 1),
      System.Globalization.NumberStyles.HexNumber);
00248                         byte b = byte.Parse(cssString.Substring(2, 1) + cssString.Substring(2, 1),
      System.Globalization.NumberStyles.HexNumber);
00249                         byte a = byte.Parse(cssString.Substring(3, 1) + cssString.Substring(3, 1),
      System.Globalization.NumberStyles.HexNumber);
00250 
00251                         return Colour.FromRgba(r, g, b, a);
00252                     }
00253                     else if (cssString.Length == 6)
00254                     {
00255                         byte r = byte.Parse(cssString.Substring(0, 2),
      System.Globalization.NumberStyles.HexNumber);
00256                         byte g = byte.Parse(cssString.Substring(2, 2),
      System.Globalization.NumberStyles.HexNumber);
00257                         byte b = byte.Parse(cssString.Substring(4, 2),
      System.Globalization.NumberStyles.HexNumber);
00258 
00259                         return Colour.FromRgb(r, g, b);
00260                     }
00261                     else if (cssString.Length == 8)
00262                     {
00263                         byte r = byte.Parse(cssString.Substring(0, 2),
      System.Globalization.NumberStyles.HexNumber);
00264                         byte g = byte.Parse(cssString.Substring(2, 2),
      System.Globalization.NumberStyles.HexNumber);
00265                         byte b = byte.Parse(cssString.Substring(4, 2),
      System.Globalization.NumberStyles.HexNumber);
00266                         byte a = byte.Parse(cssString.Substring(6, 2),
      System.Globalization.NumberStyles.HexNumber);
00267 
00268                         return Colour.FromRgba(r, g, b, a);
00269                     }
00270                     else
00271                     {
00272                         return null;
00273                     }
00274                 }
00275                 catch
00276                 {
00277                     return null;
00278                 }
00279             }
00280             else if (cssString.StartsWith("rgb(") || cssString.StartsWith("rgba("))
00281             {
00282                 try
00283                 {
00284                     cssString = cssString.Substring(cssString.IndexOf("(") + 1).Replace(")",
      "").Replace(" ", "");
00285                     string[] splitCssString = cssString.Split(',');
00286 
00287                     double R = ParseColourValueOrPercentage(splitCssString[0]);
00288                     double G = ParseColourValueOrPercentage(splitCssString[1]);
00289                     double B = ParseColourValueOrPercentage(splitCssString[2]);
00290 
00291                     double A = 1;
00292 
00293                     if (splitCssString.Length == 4)
00294                     {
```

```
00295                       A = double.Parse(splitCssString[3],
      System.Globalization.CultureInfo.InvariantCulture);
00296                   }
00297
00298                   return Colour.FromRgba(R, G, B, A);
00299               }
00300               catch
00301               {
00302                   return null;
00303               }
00304           }
00305           else
00306           {
00307               if (StandardColours.TryGetValue(cssString, out Colour tbr))
00308               {
00309                   return tbr;
00310               }
00311               else
00312               {
00313                   return null;
00314               }
00315           }
00316       }
00317
00318       private static double ParseColourValueOrPercentage(string value)
00319       {
00320           if (int.TryParse(value, out int tbr))
00321           {
00322               return tbr / 255.0;
00323           }
00324           else if (value.Contains("%"))
00325           {
00326               return double.Parse(value.Replace("%", ""),
      System.Globalization.CultureInfo.InvariantCulture) / 100.0;
00327           }
00328           else
00329           {
00330               return double.Parse(value, System.Globalization.CultureInfo.InvariantCulture);
00331           }
00332       }
00333
00334 /// <summary>
00335 /// Create a new <see cref="Colour"/> with the same RGB components as the <paramref name="original"/>
      <see cref="Colour"/>, but with the specified <paramref name="alpha"/>.
00336 /// </summary>
00337 /// <param name="original">The original <see cref="Colour"/> from which the RGB components will be
      taken.</param>
00338 /// <param name="alpha">The alpha component of the new <see cref="Colour"/>.</param>
00339 /// <returns>A <see cref="Colour"/> struct with the same RGB components as the <paramref
      name="original"/> <see cref="Colour"/> and the specified <paramref name="alpha"/>.</returns>
00340       public static Colour WithAlpha(Colour original, double alpha)
00341       {
00342           return Colour.FromRgba(original.R, original.G, original.B, alpha);
00343       }
00344
00345 /// <summary>
00346 /// Create a new <see cref="Colour"/> with the same RGB components as the <paramref name="original"/>
      <see cref="Colour"/>, but with the specified <paramref name="alpha"/>.
00347 /// </summary>
00348 /// <param name="original">The original <see cref="Colour"/> from which the RGB components will be
      taken.</param>
00349 /// <param name="alpha">The alpha component of the new <see cref="Colour"/>.</param>
00350 /// <returns>A <see cref="Colour"/> struct with the same RGB components as the <paramref
      name="original"/> <see cref="Colour"/> and the specified <paramref name="alpha"/>.</returns>
00351       public static Colour WithAlpha(Colour original, byte alpha)
00352       {
00353           return Colour.FromRgba(original.R, original.G, original.B, (double)alpha / 255.0);
00354       }
00355
00356 /// <summary>
00357 /// Create a new <see cref="Colour"/> with the same RGB components as the current <see
      cref="Colour"/>, but with the specified <paramref name="alpha"/>.
00358 /// </summary>
00359 /// <param name="alpha">The alpha component of the new <see cref="Colour"/>.</param>
00360 /// <returns>A <see cref="Colour"/> struct with the same RGB components as the current <see
      cref="Colour"/> and the specified <paramref name="alpha"/>.</returns>
00361       [Pure]
00362       public Colour WithAlpha(double alpha)
00363       {
00364           return Colour.FromRgba(this.R, this.G, this.B, alpha);
00365       }
00366
00367 /// <summary>
00368 /// Create a new <see cref="Colour"/> with the same RGB components as the current <see
      cref="Colour"/>, but with the specified <paramref name="alpha"/>.
00369 /// </summary>
00370 /// <param name="alpha">The alpha component of the new <see cref="Colour"/>.</param>
```

```
00371 /// <returns>A <see cref="Colour"/> struct with the same RGB components as the current <see
      cref="Colour"/> and the specified <paramref name="alpha"/>.</returns>
00372         [Pure]
00373         public Colour WithAlpha(byte alpha)
00374         {
00375             return Colour.FromRgba(this.R, this.G, this.B, (double)alpha / 255.0);
00376         }
00377 
00378 /// <summary>
00379 /// Converts a <see cref="Colour"/> to the CIE XYZ colour space.
00380 /// </summary>
00381 /// <returns>A <see cref="ValueTuple"/> containing the X, Y and Z components of the <see
      cref="Colour"/>.</returns>
00382         public (double X, double Y, double Z) ToXYZ()
00383         {
00384             double r, g, b;
00385 
00386             if (R <= 0.04045)
00387             {
00388                 r = 25 * R / 323;
00389             }
00390             else
00391             {
00392                 r = Math.Pow((200 * R + 11) / 211, 2.4);
00393             }
00394 
00395             if (G <= 0.04045)
00396             {
00397                 g = 25 * G / 323;
00398             }
00399             else
00400             {
00401                 g = Math.Pow((200 * G + 11) / 211, 2.4);
00402             }
00403 
00404             if (B <= 0.04045)
00405             {
00406                 b = 25 * B / 323;
00407             }
00408             else
00409             {
00410                 b = Math.Pow((200 * B + 11) / 211, 2.4);
00411             }
00412 
00413             double x = 0.41239080 * r + 0.35758434 * g + 0.18048079 * b;
00414             double y = 0.21263901 * r + 0.71516868 * g + 0.07219232 * b;
00415             double z = 0.01933082 * r + 0.11919478 * g + 0.95053215 * b;
00416 
00417             return (x, y, z);
00418         }
00419 
00420 /// <summary>
00421 /// Creates a <see cref="Colour"/> from CIE XYZ coordinates.
00422 /// </summary>
00423 /// <param name="x">The X coordinate.</param>
00424 /// <param name="y">The Y coordinate.</param>
00425 /// <param name="z">The Z coordinate.</param>
00426 /// <returns>An sRGB <see cref="Colour"/> created from the specified components.</returns>
00427         public static Colour FromXYZ(double x, double y, double z)
00428         {
00429             double r = +3.24096994 * x - 1.53738318 * y - 0.49861076 * z;
00430             double g = -0.96924364 * x + 1.8759675 * y + 0.04155506 * z;
00431             double b = 0.05563008 * x - 0.20397696 * y + 1.05697151 * z;
00432 
00433             if (r <= 0.0031308)
00434             {
00435                 r = 323 * r / 25;
00436             }
00437             else
00438             {
00439                 r = (211 * Math.Pow(r, 1 / 2.4) - 11) / 200;
00440             }
00441 
00442             if (g <= 0.0031308)
00443             {
00444                 g = 323 * g / 25;
00445             }
00446             else
00447             {
00448                 g = (211 * Math.Pow(g, 1 / 2.4) - 11) / 200;
00449             }
00450 
00451             if (b <= 0.0031308)
00452             {
00453                 b = 323 * b / 25;
00454             }
00455             else
```

```
00456             {
00457                 b = (211 * Math.Pow(b, 1 / 2.4) - 11) / 200;
00458             }
00459 
00460             r = Math.Min(Math.Max(0, r), 1);
00461             g = Math.Min(Math.Max(0, g), 1);
00462             b = Math.Min(Math.Max(0, b), 1);
00463 
00464             return Colour.FromRgb(r, g, b);
00465         }
00466 
00467 /// <summary>
00468 /// Converts a <see cref="Colour"/> to the CIE Lab colour space (under Illuminant D65).
00469 /// </summary>
00470 /// <returns>A <see cref="ValueType"/> containing the L*, a* and b* components of the <see
      cref="Colour"/>.</returns>
00471         public (double L, double a, double b) ToLab()
00472         {
00473             double f(double t)
00474             {
00475                 const double d = 6.0 / 29;
00476 
00477                 if (t > d * d * d)
00478                 {
00479                     return Math.Pow(t, 1.0 / 3);
00480                 }
00481                 else
00482                 {
00483                     return t / (3 * d * d) + 4.0 / 29;
00484                 }
00485             }
00486 
00487             const double xN = 0.950489;
00488             const double yN = 1;
00489             const double zN = 1.088840;
00490 
00491             (double x, double y, double z) = this.ToXYZ();
00492 
00493             double fY = f(y / yN);
00494 
00495             double l = 1.16 * fY - 0.16;
00496             double a = 5 * (f(x / xN) - fY);
00497             double b = 2 * (fY - f(z / zN));
00498 
00499             return (l, a, b);
00500         }
00501 
00502 /// <summary>
00503 /// Creates a <see cref="Colour"/> from CIE Lab coordinates (under Illuminant D65).
00504 /// </summary>
00505 /// <param name="L">The L* component.</param>
00506 /// <param name="a">The a* component.</param>
00507 /// <param name="b">The b* component.</param>
00508 /// <returns>An sRGB <see cref="Colour"/> created from the specified components.</returns>
00509         public static Colour FromLab(double L, double a, double b)
00510         {
00511             double f(double t)
00512             {
00513                 const double d = 6.0 / 29;
00514 
00515                 if (t > d)
00516                 {
00517                     return t * t * t;
00518                 }
00519                 else
00520                 {
00521                     return 3 * d * d * (t - 4.0 / 29);
00522                 }
00523             }
00524 
00525             const double xN = 0.950489;
00526             const double yN = 1;
00527             const double zN = 1.088840;
00528 
00529             double x = xN * f((L + 0.16) / 1.16 + a / 5);
00530             double y = yN * f((L + 0.16) / 1.16);
00531             double z = zN * f((L + 0.16) / 1.16 - b / 2);
00532 
00533             return Colour.FromXYZ(x, y, z);
00534         }
00535 
00536 /// <summary>
00537 /// Converts a <see cref="Colour"/> to the HSL colour space.
00538 /// </summary>
00539 /// <returns>A <see cref="ValueType"/> containing the H, S and L components of the <see
      cref="Colour"/>.  Each component has range [0, 1].</returns>
00540         public (double H, double S, double L) ToHSL()
```

```
00541         {
00542             double xMax = Math.Max(Math.Max(R, G), B);
00543             double xMin = Math.Min(Math.Min(R, G), B);
00544 
00545             double l = (xMax + xMin) * 0.5;
00546 
00547             double h;
00548 
00549             if (xMax == xMin)
00550             {
00551                 h = 0;
00552             }
00553             else if (xMax == R)
00554             {
00555                 h = (G - B) / (xMax - xMin) / 6;
00556             }
00557             else if (xMax == G)
00558             {
00559                 h = (2 + (B - R) / (xMax - xMin)) / 6;
00560             }
00561             else
00562             {
00563                 h = (4 + (R - G) / (xMax - xMin)) / 6;
00564             }
00565 
00566             double s;
00567 
00568             if (l == 0 || l == 1)
00569             {
00570                 s = 0;
00571             }
00572             else
00573             {
00574                 s = (xMax - l) / Math.Min(l, 1 - l);
00575             }
00576 
00577             return (h, s, l);
00578         }
00579 
00580 /// <summary>
00581 /// Creates a <see cref="Colour"/> from HSL coordinates.
00582 /// </summary>
00583 /// <param name="h">The H component.  Should be in range [0, 1].</param>
00584 /// <param name="s">The S component.  Should be in range [0, 1].</param>
00585 /// <param name="l">The L component.  Should be in range [0, 1].</param>
00586 /// <returns>A <see cref="Colour"/> created from the specified components.</returns>
00587         public static Colour FromHSL(double h, double s, double l)
00588         {
00589             double c = (1 - Math.Abs(2 * l - 1)) * s;
00590 
00591             double hp = h * 6;
00592 
00593             double x = c * (1 - Math.Abs((hp % 2) - 1));
00594 
00595             double r1, g1, b1;
00596 
00597             if (hp <= 1)
00598             {
00599                 r1 = c;
00600                 g1 = x;
00601                 b1 = 0;
00602             }
00603             else if (hp <= 2)
00604             {
00605                 r1 = x;
00606                 g1 = c;
00607                 b1 = 0;
00608             }
00609             else if (hp <= 3)
00610             {
00611                 r1 = 0;
00612                 g1 = c;
00613                 b1 = x;
00614             }
00615             else if (hp <= 4)
00616             {
00617                 r1 = 0;
00618                 g1 = x;
00619                 b1 = c;
00620             }
00621             else if (hp <= 5)
00622             {
00623                 r1 = x;
00624                 g1 = 0;
00625                 b1 = c;
00626             }
00627             else if (hp <= 6)
```

```
00628             {
00629                 r1 = c;
00630                 g1 = 0;
00631                 b1 = x;
00632             }
00633             else
00634             {
00635                 r1 = 0;
00636                 g1 = 0;
00637                 b1 = 0;
00638             }
00639 
00640             double m = l - c / 2;
00641 
00642             return Colour.FromRgb(r1 + m, g1 + m, b1 + m);
00643         }
00644 
00645         [DebuggerBrowsable(DebuggerBrowsableState.Never)]
00646         private string DebuggerDisplay
00647         {
00648             get
00649             {
00650                 return this.ToCSSString(this.A != 1);
00651             }
00652         }
00653     }
00654 }
```

8.70 Document.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Text;
00021 
00022 namespace VectSharp
00023 {
00024 /// <summary>
00025 /// Represents a collection of pages.
00026 /// </summary>
00027     public class Document
00028     {
00029 /// <summary>
00030 /// The pages in the document.
00031 /// </summary>
00032         public List<Page> Pages = new List<Page>();
00033 
00034 
00035 /// <summary>
00036 /// Create a new document.
00037 /// </summary>
00038         public Document()
00039         {
00040 
00041         }
00042     }
00043 
00044 /// <summary>
00045 /// Represents a <see cref="Graphics"/> object with a width and height.
00046 /// </summary>
00047     public class Page
00048     {
00049 /// <summary>
00050 /// Width of the page.
00051 /// </summary>
00052         public double Width { get; set; }
00053 
```

```
00054 /// <summary>
00055 /// Height of the page.
00056 /// </summary>
00057         public double Height { get; set; }
00058 
00059 /// <summary>
00060 /// Graphics surface of the page.
00061 /// </summary>
00062         public Graphics Graphics { get; set; }
00063 
00064 /// <summary>
00065 /// Background colour of the page.
00066 /// </summary>
00067         public Colour Background { get; set; } = Colour.FromRgba(255, 255, 255, 0);
00068 
00069 /// <summary>
00070 /// Create a new page.
00071 /// </summary>
00072 /// <param name="width">The width of the page.</param>
00073 /// <param name="height">The height of the page.</param>
00074         public Page(double width, double height)
00075         {
00076             this.Width = width;
00077             this.Height = height;
00078 
00079             this.Graphics = new Graphics();
00080             this.Graphics.Translate(0, 0);
00081         }
00082 
00083 /// <summary>
00084 /// Translate and resize the <see cref="Page"/> so that it displays the specified <paramref
      name="region"/>.
00085 /// </summary>
00086 /// <param name="region">The area to include in the page.</param>
00087 /// <param name="removeClippedGraphics">If this is <see langword="true"/>, graphics actions that fall
      outside of the specified region are completely removed from the plot, otherwise they are just
      hidden.</param>
00088 /// <param name="tag">A tag to identify the transform.</param>
00089         public void Crop(Rectangle region, bool removeClippedGraphics = false, string tag = null)
00090         {
00091             this.Crop(region.Location, region.Size, removeClippedGraphics, tag);
00092         }
00093 
00094 /// <summary>
00095 /// Translate and resize the <see cref="Page"/> so that it displays the rectangle defined by <paramref
      name="topLeft"/> and <paramref name="size"/>.
00096 /// </summary>
00097 /// <param name="topLeft">The top left corner of the area to include in the page.</param>
00098 /// <param name="size">The size of the area to include in the page.</param>
00099 /// <param name="removeClippedGraphics">If this is <see langword="true"/>, graphics actions that fall
      outside of the specified region are completely removed from the plot, otherwise they are just
      hidden.</param>
00100 /// <param name="tag">A tag to identify the transform.</param>
00101         public void Crop(Point topLeft, Size size, bool removeClippedGraphics = false, string tag =
      null)
00102         {
00103             if (removeClippedGraphics)
00104             {
00105                 this.Graphics.Crop(new Rectangle(topLeft, size));
00106             }
00107 
00108             if (this.Graphics.Actions[0] is TransformAction transf)
00109             {
00110                 double[,] currMatrix = transf.GetMatrix();
00111 
00112                 double[,] newMatrix = Graphics.Multiply(Graphics.TranslationMatrix(-topLeft.X,
      -topLeft.Y), currMatrix);
00113 
00114                 this.Graphics.Actions[0] = new TransformAction(newMatrix, tag);
00115             }
00116             else
00117             {
00118                 this.Graphics.Actions.Insert(0, new TransformAction(new Point(-topLeft.X, -topLeft.Y),
      tag));
00119             }
00120 
00121             this.Width = size.Width;
00122             this.Height = size.Height;
00123         }
00124 
00125 /// <summary>
00126 /// Translate and resize the <see cref="Page"/> so that it displays the rectangle corresponding to the
      bounding box of its contents.
00127 /// </summary>
00128 /// <param name="tag">A tag to identify the transform.</param>
00129         public void Crop(string tag = null)
00130         {
```

```
00131             Rectangle bounds = this.Graphics.GetBounds();
00132 
00133             if (this.Graphics.Actions[0] is TransformAction transf)
00134             {
00135                 double[,] currMatrix = transf.GetMatrix();
00136 
00137                 double[,] newMatrix = Graphics.Multiply(Graphics.TranslationMatrix(-bounds.Location.X,
      -bounds.Location.Y), currMatrix);
00138 
00139                 this.Graphics.Actions[0] = new TransformAction(newMatrix, tag);
00140             }
00141             else
00142             {
00143                 this.Graphics.Actions.Insert(0, new TransformAction(new Point(-bounds.Location.X,
      -bounds.Location.Y), tag));
00144             }
00145 
00146             this.Width = bounds.Size.Width;
00147             this.Height = bounds.Size.Height;
00148         }
00149     }
00150 }
```

8.71 Enums.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
00019 using System.Diagnostics;
00020 using System.Linq;
00021 
00022 namespace VectSharp
00023 {
00024 /// <summary>
00025 /// Represent text baselines.
00026 /// </summary>
00027     public enum TextBaselines
00028     {
00029 /// <summary>
00030 /// The current vertical coordinate determines where the top of the text string will be placed.
00031 /// </summary>
00032         Top,
00033 
00034 /// <summary>
00035 /// The current vertical coordinate determines where the bottom of the text string will be placed.
00036 /// </summary>
00037         Bottom,
00038 
00039 /// <summary>
00040 /// The current vertical coordinate determines where the middle of the text string will be placed.
00041 /// </summary>
00042         Middle,
00043 
00044 /// <summary>
00045 /// The current vertical coordinate determines where the baseline of the text string will be placed.
00046 /// </summary>
00047         Baseline
00048     }
00049 
00050 /// <summary>
00051 /// Represents text anchors.
00052 /// </summary>
00053     public enum TextAnchors
00054     {
00055 /// <summary>
00056 /// The current coordinate will determine the position of the left side of the text string.
00057 /// </summary>
00058         Left,
```

```
00059
00060 /// <summary>
00061 /// The current coordinate will determine the position of the center of the text string.
00062 /// </summary>
00063         Center,
00064
00065 /// <summary>
00066 /// The current coordinate will determine the position of the right side of the text string.
00067 /// </summary>
00068         Right
00069     }
00070
00071 /// <summary>
00072 /// Contains information about text spacing.
00073 /// </summary>
00074     public struct TextSpacing :  IEquatable<TextSpacing>
00075     {
00076 /// <summary>
00077 /// Scaling factor for whitespace characters.
00078 /// </summary>
00079         public double WhitespaceScale { get; }
00080
00081 /// <summary>
00082 /// Increment for whitespace characters.
00083 /// </summary>
00084         public double WhitespaceAdd { get; }
00085
00086
00087 /// <summary>
00088 /// Scaling factor for non-whitespace characters.
00089 /// </summary>
00090         public double NonWhitespaceScale { get; }
00091
00092 /// <summary>
00093 /// Increment for non-whitespace characters.
00094 /// </summary>
00095         public double NonWhitespaceAdd { get; }
00096
00097 /// <summary>
00098 /// Create a new <see cref="TextSpacing"/> with the specified spacing parameters.
00099 /// </summary>
00100 /// <param name="nonWhitespaceScale">Scaling factor for non-whitespace characters.</param>
00101 /// <param name="nonWhitespaceAdd">Increment for non-whitespace characters.</param>
00102 /// <param name="whitespaceScale">Scaling factor for whitespace characters.</param>
00103 /// <param name="whitespaceAdd">Increment for whitespace characters.</param>
00104         public TextSpacing(double nonWhitespaceScale, double nonWhitespaceAdd, double whitespaceScale,
      double whitespaceAdd)
00105         {
00106             this.WhitespaceScale = whitespaceScale;
00107             this.WhitespaceAdd = whitespaceAdd;
00108             this.NonWhitespaceScale = nonWhitespaceScale;
00109             this.NonWhitespaceAdd = nonWhitespaceAdd;
00110         }
00111
00112 /// <summary>
00113 /// Create a new <see cref="TextSpacing"/> with the specified spacing parameters.
00114 /// </summary>
00115 /// <param name="scale">Scaling factor.</param>
00116 /// <param name="add">Increment.</param>
00117         public TextSpacing(double scale, double add)
00118         {
00119             this.WhitespaceScale = scale;
00120             this.WhitespaceAdd = add;
00121             this.NonWhitespaceScale = scale;
00122             this.NonWhitespaceAdd = add;
00123         }
00124
00125 /// <summary>
00126 /// Default text spacing.
00127 /// </summary>
00128         public static TextSpacing Default { get; } = new TextSpacing(1, 0);
00129
00130 /// <inheritdoc/>
00131         public bool Equals(TextSpacing other)
00132         {
00133             return this.WhitespaceScale == other.WhitespaceScale && this.WhitespaceAdd ==
      other.WhitespaceAdd && this.NonWhitespaceScale == other.NonWhitespaceScale && this.NonWhitespaceAdd ==
      other.NonWhitespaceAdd;
00134         }
00135
00136 /// <inheritdoc/>
00137         public override bool Equals(object obj)
00138         {
00139             if (obj is TextSpacing other)
00140             {
00141                 return Equals(other);
00142             }
```

```
00143             else
00144             {
00145                 return false;
00146             }
00147         }
00148 
00149 /// <inheritdoc/>
00150         public override int GetHashCode()
00151         {
00152             unchecked
00153             {
00154                 int hash = 17;
00155                 hash = hash * 31 + WhitespaceScale.GetHashCode();
00156                 hash = hash * 31 + WhitespaceAdd.GetHashCode();
00157                 hash = hash * 31 + NonWhitespaceAdd.GetHashCode();
00158                 hash = hash * 31 + NonWhitespaceAdd.GetHashCode();
00159                 return hash;
00160             }
00161         }
00162 
00163 /// <summary>
00164 /// Compares two <see cref="TextSpacing"/> objects.
00165 /// </summary>
00166 /// <param name="left">The first <see cref="TextSpacing"/> object.</param>
00167 /// <param name="right">The second <see cref="TextSpacing"/> object.</param>
00168 /// <returns><see langword="true"/> if the two <see cref="TextSpacing"/> objects specify the same
      spacing parameters, <see langword="false"/> otherwise.</returns>
00169         public static bool operator ==(TextSpacing left, TextSpacing right)
00170         {
00171             return Equals(left, right);
00172         }
00173 
00174 /// <summary>
00175 /// Compares two <see cref="TextSpacing"/> objects.
00176 /// </summary>
00177 /// <param name="left">The first <see cref="TextSpacing"/> object.</param>
00178 /// <param name="right">The second <see cref="TextSpacing"/> object.</param>
00179 /// <returns><see langword="false"/> if the two <see cref="TextSpacing"/> objects specify the same
      spacing parameters, <see langword="true"/> otherwise.</returns>
00180         public static bool operator !=(TextSpacing left, TextSpacing right)
00181         {
00182             return !Equals(left, right);
00183         }
00184     }
00185 
00186 /// <summary>
00187 /// Represents line caps.
00188 /// </summary>
00189     public enum LineCaps
00190     {
00191 /// <summary>
00192 /// The ends of the line are squared off at the endpoints.
00193 /// </summary>
00194         Butt = 0,
00195 
00196 /// <summary>
00197 /// The ends of the lines are rounded.
00198 /// </summary>
00199         Round = 1,
00200 
00201 /// <summary>
00202 /// The ends of the lines are squared off by adding an half square box at each end.
00203 /// </summary>
00204         Square = 2
00205     }
00206 
00207 /// <summary>
00208 /// Represents line joining options.
00209 /// </summary>
00210     public enum LineJoins
00211     {
00212 /// <summary>
00213 /// Consecutive segments are joined by straight corners.
00214 /// </summary>
00215         Bevel = 2,
00216 
00217 /// <summary>
00218 /// Consecutive segments are joined by extending their outside edges until they meet.
00219 /// </summary>
00220         Miter = 0,
00221 
00222 /// <summary>
00223 /// Consecutive segments are joined by arc segments.
00224 /// </summary>
00225         Round = 1
00226     }
00227 
```

```
00228 /// <summary>
00229 /// Represents instructions on how to paint a dashed line.
00230 /// </summary>
00231     [DebuggerDisplay("{DebuggerDisplay,nq}")]
00232     public struct LineDash
00233     {
00234 /// <summary>
00235 /// A solid (not dashed) line
00236 /// </summary>
00237         public static LineDash SolidLine = new LineDash(0, 0, 0);
00238 
00239 /// <summary>
00240 /// Length of the "on" (painted) segment.
00241 /// </summary>
00242         [Obsolete("Please use the LineDash.DashArray instead.")]
00243         public double UnitsOn;
00244 
00245 /// <summary>
00246 /// Length of the "off" (not painted) segment.
00247 /// </summary>
00248         [Obsolete("Please use the LineDash.DashArray instead.")]
00249         public double UnitsOff;
00250 
00251 /// <summary>
00252 /// An array specifying lenghts of alternating dashes and gaps.
00253 /// </summary>
00254         public double[] DashArray;
00255 
00256 /// <summary>
00257 /// Position in the dash pattern at which the line starts.
00258 /// </summary>
00259         public double Phase;
00260 
00261 /// <summary>
00262 /// Define a new line dash pattern.
00263 /// </summary>
00264 /// <param name="unitsOn">The length of the "on" (painted) segment.</param>
00265 /// <param name="unitsOff">The length of the "off" (not painted) segment.</param>
00266 /// <param name="phase">The position in the dash pattern at which the line starts.</param>
00267         public LineDash(double unitsOn, double unitsOff, double phase)
00268         {
00269             this.DashArray = new double[] { unitsOn, unitsOff };
00270             Phase = phase;
00271 
00272             /* Deprecated */
00273 #pragma warning disable 618
00274             this.UnitsOn = unitsOn;
00275             this.UnitsOff = unitsOff;
00276 #pragma warning restore 618
00277         }
00278 
00279 /// <summary>
00280 /// Define a new line dash pattern.
00281 /// </summary>
00282 /// <param name="units">The length of the dash segments and gaps.</param>
00283 /// <param name="phase">The position in the dash pattern at which the line starts.</param>
00284         public LineDash(double units, double phase = 0)
00285         {
00286             this.DashArray = new double[] { units };
00287             Phase = phase;
00288 
00289             /* Deprecated */
00290 #pragma warning disable 618
00291             this.UnitsOn = units;
00292             this.UnitsOff = units;
00293 #pragma warning restore 618
00294         }
00295 
00296 /// <summary>
00297 /// Define a new line dash pattern.
00298 /// </summary>
00299 /// <param name="dashArray">An array specifying lenghts of alternating dashes and gaps.</param>
00300 /// <param name="phase">The position in the dash pattern at which the line starts.</param>
00301         public LineDash(double[] dashArray, double phase = 0)
00302         {
00303             this.DashArray = dashArray;
00304             this.Phase = phase;
00305 
00306             /* Deprecated */
00307 #pragma warning disable 618
00308             this.UnitsOn = dashArray != null && dashArray.Length > 0 ?  dashArray[0] :  0;
00309             this.UnitsOff = dashArray != null && dashArray.Length > 0 ?
      dashArray[System.Math.Min(dashArray.Length - 1, 1)] :  0;
00310 #pragma warning restore 618
00311         }
00312 
00313         [DebuggerBrowsable(DebuggerBrowsableState.Never)]
```

```
00314         private string DebuggerDisplay
00315         {
00316             get
00317             {
00318                 if (this.Phase == 0 && this.DashArray.All(x => x == 0))
00319                 {
00320                     return "{ Solid line }";
00321                 }
00322                 else
00323                 {
00324                     return string.Format("{{ Phase:  {0}, DashArray:  {1} }}", this.Phase, "[ " +
      this.DashArray.Take(Math.Min(4, this.DashArray.Length)).Select(x => x.ToString()).Aggregate((a, b) =>
      a + ", " + b) + (this.DashArray.Length > 4 ?  " ..." :  "") + " ]");
00325                 }
00326             }
00327         }
00328     }
00329
00330 /// <summary>
00331 /// Types of <see cref="Segment"/>.
00332 /// </summary>
00333     public enum SegmentType
00334     {
00335 /// <summary>
00336 /// The segment represents a move from the current point to a new point.
00337 /// </summary>
00338         Move,
00339
00340 /// <summary>
00341 /// The segment represents a straight line from the current point to a new point.
00342 /// </summary>
00343         Line,
00344
00345 /// <summary>
00346 /// The segment represents a cubic bezier curve from the current point to a new point.
00347 /// </summary>
00348         CubicBezier,
00349
00350 /// <summary>
00351 /// The segment represents a circular arc from the current point to a new point.
00352 /// </summary>
00353         Arc,
00354
00355 /// <summary>
00356 /// The segment represents the closing segment of a figure.
00357 /// </summary>
00358         Close
00359     }
00360
00361 /// <summary>
00362 /// Represents ways to deal with unbalanced graphics state stacks.
00363 /// </summary>
00364     public enum UnbalancedStackActions
00365     {
00366 /// <summary>
00367 /// If the graphics state stack is unbalanced, an exception will be thrown.
00368 /// </summary>
00369         Throw,
00370
00371 /// <summary>
00372 /// The graphics state stack will be automatically balanced by adding or removing calls to <see
      cref="Graphics.Restore"/> as necessary.
00373 /// </summary>
00374         SilentlyFix,
00375
00376 /// <summary>
00377 /// No attempt will be made at correcting an unbalanced graphics state stack.  This may cause issues
      with some consumers.
00378 /// </summary>
00379         Ignore
00380     }
00381 }
```

8.72 BoxBlurFilter.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
```

```
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Diagnostics.Contracts;
00020 using System.Threading.Tasks;
00021
00022 namespace VectSharp.Filters
00023 {
00024 /// <summary>
00025 /// Represents a filter applying a box blur.
00026 /// </summary>
00027     public class BoxBlurFilter :  ILocationInvariantFilter
00028     {
00029 /// <summary>
00030 /// The radius of the box blur (the actual size of the box is 2 * <see cref="BoxRadius"/> + 1).
00031 /// </summary>
00032         public double BoxRadius { get; }
00033
00034 /// <inheritdoc/>
00035         public Point TopLeftMargin { get; }
00036 /// <inheritdoc/>
00037         public Point BottomRightMargin { get; }
00038
00039 /// <summary>
00040 /// Creates a new <see cref="BoxBlurFilter"/> with the specified radius.
00041 /// </summary>
00042 /// <param name="boxRadius">The radius of the box blur (the actual size of the box is 2 * <paramref
      name="boxRadius"/> + 1).</param>
00043         public BoxBlurFilter(double boxRadius)
00044         {
00045             this.BoxRadius = boxRadius;
00046             this.TopLeftMargin = new Point(boxRadius, boxRadius);
00047             this.BottomRightMargin = new Point(boxRadius, boxRadius);
00048         }
00049
00050 /// <inheritdoc/>
00051         [Pure]
00052         public RasterImage Filter(RasterImage image, double scale)
00053         {
00054             return BoxBlurSRGB(image, (int)Math.Round(this.BoxRadius * scale));
00055         }
00056
00057         private RasterImage BoxBlurSRGB(RasterImage image, int boxRadius)
00058         {
00059             IntPtr intermediateData = System.Runtime.InteropServices.Marshal.AllocHGlobal(image.Width
      * image.Height * (image.HasAlpha ?  4 :  3));
00060             IntPtr tbrData = System.Runtime.InteropServices.Marshal.AllocHGlobal(image.Width *
      image.Height * (image.HasAlpha ?  4 :  3));
00061             GC.AddMemoryPressure(2 * image.Width * image.Height * (image.HasAlpha ?  4 :  3));
00062
00063             int width = image.Width;
00064             int height = image.Height;
00065
00066             int pixelSize = image.HasAlpha ?  4 :  3;
00067             int stride = image.Width * pixelSize;
00068
00069             double normFactor = 1.0 / (2 * boxRadius + 1);
00070
00071             int threads;
00072
00073             double size = Math.Sqrt((double)image.Width * image.Height);
00074
00075             if (size <= 128)
00076             {
00077                 threads = 1;
00078             }
00079             else if (size <= 512)
00080             {
00081                 threads = Math.Min(4, Environment.ProcessorCount);
00082             }
00083             else
00084             {
00085                 threads = Math.Min(8, Environment.ProcessorCount);
00086             }
00087
00088             unsafe
00089             {
00090                 byte* input = (byte*)image.ImageDataAddress;
00091                 byte* intermediate = (byte*)intermediateData;
00092                 byte* output = (byte*)tbrData;
```

```
00093
00094                 Action<int> yLoop;
00095
00096                 if (image.HasAlpha)
00097                 {
00098                     yLoop = y =>
00099                     {
00100                         double rAcc = 0;
00101                         double gAcc = 0;
00102                         double bAcc = 0;
00103                         double aAcc = 0;
00104
00105                         double prevR = 0;
00106                         double prevG = 0;
00107                         double prevB = 0;
00108                         double prevA = 0;
00109
00110                         double r = 0;
00111                         double g = 0;
00112                         double b = 0;
00113
00114                         double a = input[y * stride + 3] / 255.0;
00115
00116                         prevR = input[y * stride] * a;
00117                         prevG = input[y * stride + 1] * a;
00118                         prevB = input[y * stride + 2] * a;
00119                         prevA = a;
00120
00121                         rAcc = prevR * (boxRadius + 1);
00122                         gAcc = prevG * (boxRadius + 1);
00123                         bAcc = prevB * (boxRadius + 1);
00124                         aAcc = a * (boxRadius + 1);
00125
00126                         int rX = 0;
00127
00128                         for (int x = 0; x < boxRadius; x++)
00129                         {
00130                             rX = Math.Min(x, width - 1);
00131
00132                             a = input[y * stride + rX * 4 + 3] / 255.0;
00133
00134                             rAcc += input[y * stride + rX * 4] * a;
00135                             gAcc += input[y * stride + rX * 4 + 1] * a;
00136                             bAcc += input[y * stride + rX * 4 + 2] * a;
00137                             aAcc += a;
00138                         }
00139
00140                         rX = Math.Min(boxRadius, width - 1);
00141                         int lX = 0;
00142
00143                         for (int x = 0; x < width; x++)
00144                         {
00145                             rX = Math.Min(x + boxRadius, width - 1);
00146                             lX = Math.Max(x - boxRadius - 1, 0);
00147
00148                             a = input[y * stride + rX * 4 + 3] / 255.0;
00149                             r = input[y * stride + rX * 4] * a;
00150                             g = input[y * stride + rX * 4 + 1] * a;
00151                             b = input[y * stride + rX * 4 + 2] * a;
00152
00153                             prevA = input[y * stride + lX * 4 + 3] / 255.0;
00154                             prevR = input[y * stride + lX * 4] * prevA;
00155                             prevG = input[y * stride + lX * 4 + 1] * prevA;
00156                             prevB = input[y * stride + lX * 4 + 2] * prevA;
00157
00158                             rAcc += r - prevR;
00159                             gAcc += g - prevG;
00160                             bAcc += b - prevB;
00161                             aAcc += a - prevA;
00162
00163                             if (aAcc != 0)
00164                             {
00165                                 intermediate[y * stride + x * 4] = (byte)Math.Min(255, Math.Max(0,
      rAcc / aAcc));
00166                                 intermediate[y * stride + x * 4 + 1] = (byte)Math.Min(255, Math.Max(0,
      gAcc / aAcc));
00167                                 intermediate[y * stride + x * 4 + 2] = (byte)Math.Min(255, Math.Max(0,
      bAcc / aAcc));
00168                                 intermediate[y * stride + x * 4 + 3] = (byte)Math.Min(255, Math.Max(0,
      aAcc * 255 * normFactor));
00169                             }
00170                             else
00171                             {
00172                                 intermediate[y * stride + x * 4] = 0;
00173                                 intermediate[y * stride + x * 4 + 1] = 0;
00174                                 intermediate[y * stride + x * 4 + 2] = 0;
00175                                 intermediate[y * stride + x * 4 + 3] = 0;
```

```
00176                             }
00177                         }
00178                     };
00179                 }
00180                 else
00181                 {
00182                     yLoop = y =>
00183                     {
00184                         double rAcc = 0;
00185                         double gAcc = 0;
00186                         double bAcc = 0;
00187 
00188                         double prevR = 0;
00189                         double prevG = 0;
00190                         double prevB = 0;
00191 
00192                         double r = 0;
00193                         double g = 0;
00194                         double b = 0;
00195 
00196                         prevR = input[y * stride];
00197                         prevG = input[y * stride + 1];
00198                         prevB = input[y * stride + 2];
00199 
00200                         rAcc = prevR * (boxRadius + 1);
00201                         gAcc = prevG * (boxRadius + 1);
00202                         bAcc = prevB * (boxRadius + 1);
00203 
00204                         int rX = 0;
00205 
00206                         for (int x = 0; x < boxRadius; x++)
00207                         {
00208                             rX = Math.Min(x, width - 1);
00209 
00210                             rAcc += input[y * stride + rX * 3];
00211                             gAcc += input[y * stride + rX * 3 + 1];
00212                             bAcc += input[y * stride + rX * 3 + 2];
00213                         }
00214 
00215                         rX = Math.Min(boxRadius, width - 1);
00216                         int lX = 0;
00217 
00218                         for (int x = 0; x < width; x++)
00219                         {
00220                             rX = Math.Min(x + boxRadius, width - 1);
00221                             lX = Math.Max(x - boxRadius - 1, 0);
00222 
00223                             r = input[y * stride + rX * 3];
00224                             g = input[y * stride + rX * 3 + 1];
00225                             b = input[y * stride + rX * 3 + 2];
00226 
00227                             prevR = input[y * stride + lX * 3];
00228                             prevG = input[y * stride + lX * 3 + 1];
00229                             prevB = input[y * stride + lX * 3 + 2];
00230 
00231                             rAcc += r - prevR;
00232                             gAcc += g - prevG;
00233                             bAcc += b - prevB;
00234 
00235                             intermediate[y * stride + x * 3] = (byte)Math.Min(255, Math.Max(0, rAcc *
      normFactor));
00236                             intermediate[y * stride + x * 3 + 1] = (byte)Math.Min(255, Math.Max(0,
      gAcc * normFactor));
00237                             intermediate[y * stride + x * 3 + 2] = (byte)Math.Min(255, Math.Max(0,
      bAcc * normFactor));
00238                         }
00239                     };
00240                 }
00241 
00242                 Action<int> xLoop;
00243 
00244                 if (image.HasAlpha)
00245                 {
00246                     xLoop = x =>
00247                     {
00248                         double rAcc = 0;
00249                         double gAcc = 0;
00250                         double bAcc = 0;
00251                         double aAcc = 0;
00252 
00253                         double prevR = 0;
00254                         double prevG = 0;
00255                         double prevB = 0;
00256                         double prevA = 0;
00257 
00258                         double r = 0;
00259                         double g = 0;
```

```
00260                           double b = 0;
00261
00262                           double a = intermediate[x * 4 + 3] / 255.0;
00263
00264                           prevR = intermediate[x * 4] * a;
00265                           prevG = intermediate[x * 4 + 1] * a;
00266                           prevB = intermediate[x * 4 + 2] * a;
00267                           prevA = a;
00268
00269                           rAcc = prevR * (boxRadius + 1);
00270                           gAcc = prevG * (boxRadius + 1);
00271                           bAcc = prevB * (boxRadius + 1);
00272                           aAcc = a * (boxRadius + 1);
00273
00274                           int rY = 0;
00275
00276                           for (int y = 0; y < boxRadius; y++)
00277                           {
00278                               rY = Math.Min(y, height - 1);
00279
00280                               a = intermediate[rY * stride + x * 4 + 3] / 255.0;
00281
00282                               rAcc += intermediate[rY * stride + x * 4] * a;
00283                               gAcc += intermediate[rY * stride + x * 4 + 1] * a;
00284                               bAcc += intermediate[rY * stride + x * 4 + 2] * a;
00285                               aAcc += a;
00286                           }
00287
00288                           rY = Math.Min(boxRadius, height - 1);
00289                           int lY = 0;
00290
00291                           for (int y = 0; y < height; y++)
00292                           {
00293                               rY = Math.Min(y + boxRadius, height - 1);
00294                               lY = Math.Max(y - boxRadius - 1, 0);
00295
00296                               a = intermediate[rY * stride + x * 4 + 3] / 255.0;
00297                               r = intermediate[rY * stride + x * 4] * a;
00298                               g = intermediate[rY * stride + x * 4 + 1] * a;
00299                               b = intermediate[rY * stride + x * 4 + 2] * a;
00300
00301                               prevA = intermediate[lY * stride + x * 4 + 3] / 255.0;
00302                               prevR = intermediate[lY * stride + x * 4] * prevA;
00303                               prevG = intermediate[lY * stride + x * 4 + 1] * prevA;
00304                               prevB = intermediate[lY * stride + x * 4 + 2] * prevA;
00305
00306                               rAcc += r - prevR;
00307                               gAcc += g - prevG;
00308                               bAcc += b - prevB;
00309                               aAcc += a - prevA;
00310
00311                               if (aAcc != 0)
00312                               {
00313                                   output[y * stride + x * 4] = (byte)Math.Min(255, Math.Max(0, rAcc /
      aAcc));
00314                                   output[y * stride + x * 4 + 1] = (byte)Math.Min(255, Math.Max(0, gAcc
      / aAcc));
00315                                   output[y * stride + x * 4 + 2] = (byte)Math.Min(255, Math.Max(0, bAcc
      / aAcc));
00316                                   output[y * stride + x * 4 + 3] = (byte)Math.Min(255, Math.Max(0, aAcc
      * 255 * normFactor));
00317                               }
00318                               else
00319                               {
00320                                   output[y * stride + x * 4] = 0;
00321                                   output[y * stride + x * 4 + 1] = 0;
00322                                   output[y * stride + x * 4 + 2] = 0;
00323                                   output[y * stride + x * 4 + 3] = 0;
00324                               }
00325                           }
00326                       };
00327                   }
00328                   else
00329                   {
00330                       xLoop = x =>
00331                       {
00332                           double rAcc = 0;
00333                           double gAcc = 0;
00334                           double bAcc = 0;
00335
00336                           double prevR = 0;
00337                           double prevG = 0;
00338                           double prevB = 0;
00339
00340                           double r = 0;
00341                           double g = 0;
00342                           double b = 0;
```

```
00343
00344                           prevR = intermediate[x * 3];
00345                           prevG = intermediate[x * 3 + 1];
00346                           prevB = intermediate[x * 3 + 2];
00347
00348                           rAcc = prevR * (boxRadius + 1);
00349                           gAcc = prevG * (boxRadius + 1);
00350                           bAcc = prevB * (boxRadius + 1);
00351
00352                           int rY = 0;
00353
00354                           for (int y = 0; y < boxRadius; y++)
00355                           {
00356                               rY = Math.Min(y, height - 1);
00357
00358                               rAcc += intermediate[rY * stride + x * 3];
00359                               gAcc += intermediate[rY * stride + x * 3 + 1];
00360                               bAcc += intermediate[rY * stride + x * 3 + 2];
00361                           }
00362
00363                           rY = Math.Min(boxRadius, height - 1);
00364                           int lY = 0;
00365
00366                           for (int y = 0; y < height; y++)
00367                           {
00368                               rY = Math.Min(y + boxRadius, height - 1);
00369                               lY = Math.Max(y - boxRadius - 1, 0);
00370
00371                               r = intermediate[rY * stride + x * 3];
00372                               g = intermediate[rY * stride + x * 3 + 1];
00373                               b = intermediate[rY * stride + x * 3 + 2];
00374
00375                               prevR = intermediate[lY * stride + x * 3];
00376                               prevG = intermediate[lY * stride + x * 3 + 1];
00377                               prevB = intermediate[lY * stride + x * 3 + 2];
00378
00379                               rAcc += r - prevR;
00380                               gAcc += g - prevG;
00381                               bAcc += b - prevB;
00382
00383                               output[y * stride + x * 3] = (byte)Math.Min(255, Math.Max(0, rAcc *
      normFactor));
00384                               output[y * stride + x * 3 + 1] = (byte)Math.Min(255, Math.Max(0, gAcc *
      normFactor));
00385                               output[y * stride + x * 3 + 2] = (byte)Math.Min(255, Math.Max(0, bAcc *
      normFactor));
00386                           }
00387                       };
00388                   }
00389
00390                   if (threads == 1)
00391                   {
00392                       for (int y = 0; y < height; y++)
00393                       {
00394                           yLoop(y);
00395                       }
00396
00397                       for (int x = 0; x < width; x++)
00398                       {
00399                           xLoop(x);
00400                       }
00401                   }
00402                   else
00403                   {
00404                       ParallelOptions options = new ParallelOptions() { MaxDegreeOfParallelism = threads
      };
00405
00406                       Parallel.For(0, height, options, yLoop);
00407                       Parallel.For(0, width, options, xLoop);
00408                   }
00409               }
00410
00411               System.Runtime.InteropServices.Marshal.FreeHGlobal(intermediateData);
00412               GC.RemoveMemoryPressure(image.Width * image.Height * (image.HasAlpha ?  4 :  3));
00413
00414               DisposableIntPtr disp = new DisposableIntPtr(tbrData);
00415
00416               return new RasterImage(ref disp, image.Width, image.Height, image.HasAlpha,
      image.Interpolate);
00417           }
00418       }
00419 }
```

8.73 ColourMatrixFilter.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Diagnostics.Contracts;
00020 using System.Threading.Tasks;
00021
00022 namespace VectSharp.Filters
00023 {
00024 /// <summary>
00025 /// Represents a colour transformation matrix.
00026 /// </summary>
00027     public class ColourMatrix
00028     {
00029 /// <summary>
00030 /// The coefficient relating the R component of the output colour to the R component of the input
      colour.
00031 /// </summary>
00032         public double R1 { get; set; }
00033
00034 /// <summary>
00035 /// The coefficient relating the R component of the output colour to the G component of the input
      colour.
00036 /// </summary>
00037         public double R2 { get; set; }
00038
00039 /// <summary>
00040 /// The coefficient relating the R component of the output colour to the B component of the input
      colour.
00041 /// </summary>
00042         public double R3 { get; set; }
00043
00044 /// <summary>
00045 /// The coefficient relating the R component of the output colour to the A component of the input
      colour.
00046 /// </summary>
00047         public double R4 { get; set; }
00048
00049 /// <summary>
00050 /// The bias (translation) applied to the R component of the output colour.
00051 /// </summary>
00052         public double R5 { get; set; }
00053
00054 /// <summary>
00055 /// The coefficient relating the G component of the output colour to the R component of the input
      colour.
00056 /// </summary>
00057         public double G1 { get; set; }
00058
00059 /// <summary>
00060 /// The coefficient relating the G component of the output colour to the G component of the input
      colour.
00061 /// </summary>
00062         public double G2 { get; set; }
00063
00064 /// <summary>
00065 /// The coefficient relating the G component of the output colour to the B component of the input
      colour.
00066 /// </summary>
00067         public double G3 { get; set; }
00068
00069 /// <summary>
00070 /// The coefficient relating the G component of the output colour to the A component of the input
      colour.
00071 /// </summary>
00072         public double G4 { get; set; }
00073
00074 /// <summary>
00075 /// The bias (translation) applied to the R component of the output colour.
00076 /// </summary>
00077         public double G5 { get; set; }
```

```
00078
00079 /// <summary>
00080 /// The coefficient relating the B component of the output colour to the R component of the input
      colour.
00081 /// </summary>
00082         public double B1 { get; set; }
00083
00084 /// <summary>
00085 /// The coefficient relating the B component of the output colour to the G component of the input
      colour.
00086 /// </summary>
00087         public double B2 { get; set; }
00088
00089 /// <summary>
00090 /// The coefficient relating the B component of the output colour to the B component of the input
      colour.
00091 /// </summary>
00092         public double B3 { get; set; }
00093
00094 /// <summary>
00095 /// The coefficient relating the B component of the output colour to the A component of the input
      colour.
00096 /// </summary>
00097         public double B4 { get; set; }
00098
00099 /// <summary>
00100 /// The bias (translation) applied to the B component of the output colour.
00101 /// </summary>
00102         public double B5 { get; set; }
00103
00104 /// <summary>
00105 /// The coefficient relating the A component of the output colour to the R component of the input
      colour.
00106 /// </summary>
00107         public double A1 { get; set; }
00108
00109 /// <summary>
00110 /// The coefficient relating the A component of the output colour to the G component of the input
      colour.
00111 /// </summary>
00112         public double A2 { get; set; }
00113
00114 /// <summary>
00115 /// The coefficient relating the A component of the output colour to the B component of the input
      colour.
00116 /// </summary>
00117         public double A3 { get; set; }
00118
00119 /// <summary>
00120 /// The coefficient relating the A component of the output colour to the A component of the input
      colour.
00121 /// </summary>
00122         public double A4 { get; set; }
00123
00124 /// <summary>
00125 /// The bias (translation) applied to the A component of the output colour.
00126 /// </summary>
00127         public double A5 { get; set; }
00128
00129 /// <summary>
00130 /// Gets or sets the requested element of the matrix.  Elements of the last row of the matrix can be
      read, but not set.
00131 /// </summary>
00132 /// <param name="y">The row of the matrix.</param>
00133 /// <param name="x">The column of the matrix.</param>
00134 /// <returns>The requested element of the matrix.</returns>
00135 /// <exception cref="ArgumentOutOfRangeException">An attempt has been made to access an element out of
      the bounds of the matrix.</exception>
00136         public double this[int y, int x]
00137         {
00138             get
00139             {
00140                 switch (y)
00141                 {
00142                     case 0:
00143                         switch (x)
00144                         {
00145                             case 0:
00146                                 return R1;
00147                             case 1:
00148                                 return R2;
00149                             case 2:
00150                                 return R3;
00151                             case 3:
00152                                 return R4;
00153                             case 4:
00154                                 return R5;
```

```
00155                               default:
00156                                   throw new ArgumentOutOfRangeException(nameof(x), x, "Coordinate out of
      range!");
00157                           }
00158
00159                       case 1:
00160                           switch (x)
00161                           {
00162                               case 0:
00163                                   return G1;
00164                               case 1:
00165                                   return G2;
00166                               case 2:
00167                                   return G3;
00168                               case 3:
00169                                   return G4;
00170                               case 4:
00171                                   return G5;
00172                               default:
00173                                   throw new ArgumentOutOfRangeException(nameof(x), x, "Coordinate out of
      range!");
00174                           }
00175
00176                       case 2:
00177                           switch (x)
00178                           {
00179                               case 0:
00180                                   return B1;
00181                               case 1:
00182                                   return B2;
00183                               case 2:
00184                                   return B3;
00185                               case 3:
00186                                   return B4;
00187                               case 4:
00188                                   return B5;
00189                               default:
00190                                   throw new ArgumentOutOfRangeException(nameof(x), x, "Coordinate out of
      range!");
00191                           }
00192
00193                       case 3:
00194                           switch (x)
00195                           {
00196                               case 0:
00197                                   return A1;
00198                               case 1:
00199                                   return A2;
00200                               case 2:
00201                                   return A3;
00202                               case 3:
00203                                   return A4;
00204                               case 4:
00205                                   return A5;
00206                               default:
00207                                   throw new ArgumentOutOfRangeException(nameof(x), x, "Coordinate out of
      range!");
00208                           }
00209
00210                       case 4:
00211                           switch (x)
00212                           {
00213                               case 0:
00214                                   return 0;
00215                               case 1:
00216                                   return 0;
00217                               case 2:
00218                                   return 0;
00219                               case 3:
00220                                   return 0;
00221                               case 4:
00222                                   return 1;
00223                               default:
00224                                   throw new ArgumentOutOfRangeException(nameof(x), x, "Coordinate out of
      range!");
00225                           }
00226
00227                       default:
00228                           throw new ArgumentOutOfRangeException(nameof(y), y, "Coordinate out of
      range!");
00229                   }
00230               }
00231
00232               private set
00233               {
00234                   switch (y)
00235                   {
```

```
00236                     case 0:
00237                         switch (x)
00238                         {
00239                             case 0:
00240                                 R1 = value;
00241                                 break;
00242                             case 1:
00243                                 R2 = value;
00244                                 break;
00245                             case 2:
00246                                 R3 = value;
00247                                 break;
00248                             case 3:
00249                                 R4 = value;
00250                                 break;
00251                             case 4:
00252                                 R5 = value;
00253                                 break;
00254                             default:
00255                                 throw new ArgumentOutOfRangeException(nameof(x), x, "Coordinate out of
      range!");
00256                         }
00257                         break;
00258 
00259                     case 1:
00260                         switch (x)
00261                         {
00262                             case 0:
00263                                 G1 = value;
00264                                 break;
00265                             case 1:
00266                                 G2 = value;
00267                                 break;
00268                             case 2:
00269                                 G3 = value;
00270                                 break;
00271                             case 3:
00272                                 G4 = value;
00273                                 break;
00274                             case 4:
00275                                 G5 = value;
00276                                 break;
00277                             default:
00278                                 throw new ArgumentOutOfRangeException(nameof(x), x, "Coordinate out of
      range!");
00279                         }
00280                         break;
00281 
00282                     case 2:
00283                         switch (x)
00284                         {
00285                             case 0:
00286                                 B1 = value;
00287                                 break;
00288                             case 1:
00289                                 B2 = value;
00290                                 break;
00291                             case 2:
00292                                 B3 = value;
00293                                 break;
00294                             case 3:
00295                                 B4 = value;
00296                                 break;
00297                             case 4:
00298                                 B5 = value;
00299                                 break;
00300                             default:
00301                                 throw new ArgumentOutOfRangeException(nameof(x), x, "Coordinate out of
      range!");
00302                         }
00303                         break;
00304 
00305                     case 3:
00306                         switch (x)
00307                         {
00308                             case 0:
00309                                 A1 = value;
00310                                 break;
00311                             case 1:
00312                                 A2 = value;
00313                                 break;
00314                             case 2:
00315                                 A3 = value;
00316                                 break;
00317                             case 3:
00318                                 A4 = value;
00319                                 break;
```

```
00320                               case 4:
00321                                   A5 = value;
00322                                   break;
00323                               default:
00324                                   throw new ArgumentOutOfRangeException(nameof(x), x, "Coordinate out of
      range!");
00325                           }
00326                           break;
00327
00328                       default:
00329                           throw new ArgumentOutOfRangeException(nameof(y), y, "Coordinate out of
      range!");
00330                   }
00331               }
00332           }
00333
00334 /// <summary>
00335 /// A <see cref="ColourMatrix"/> that whose output colour is always the same as the input colour.
00336 /// </summary>
00337           public static ColourMatrix Identity = new ColourMatrix(new double[,] { { 1, 0, 0, 0, 0 }, { 0,
      1, 0, 0, 0 }, { 0, 0, 1, 0, 0 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } });
00338
00339 /// <summary>
00340 /// A <see cref="ColourMatrix"/> that transforms every colour in a shade of grey with approximately
      the same luminance.
00341 /// </summary>
00342           public static ColourMatrix GreyScale = new ColourMatrix(new double[,] { { 0.2126, 0.7152,
      0.0722, 0, 0 }, { 0.2126, 0.7152, 0.0722, 0, 0 }, { 0.2126, 0.7152, 0.0722, 0, 0 }, { 0, 0, 0, 1, 0 },
      { 0, 0, 0, 0, 1 } });
00343
00344 /// <summary>
00345 /// A <see cref="ColourMatrix"/> producing a "pastel" (desaturation) effect.
00346 /// </summary>
00347           public static ColourMatrix Pastel = new ColourMatrix(new double[,] { { 0.75, 0.25, 0.25, 0, 0
      }, { 0.25, 0.75, 0.25, 0, 0 }, { 0.25, 0.25, 0.75, 0, 0 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } });
00348
00349 /// <summary>
00350 /// A <see cref="ColourMatrix"/> that inverts every colour, leaving the alpha component intact.
00351 /// </summary>
00352           public static ColourMatrix Inversion = new ColourMatrix(new double[,] { { -1, 0, 0, 0, 1 }, {
      0, -1, 0, 0, 1 }, { 0, 0, -1, 0, 1 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } });
00353
00354 /// <summary>
00355 /// A <see cref="ColourMatrix"/> that inverts the alpha component, leaving the other components
      intact.
00356 /// </summary>
00357           public static ColourMatrix AlphaInversion = new ColourMatrix(new double[,] { { 1, 0, 0, 0, 0
      }, { 0, 1, 0, 0, 0 }, { 0, 0, 1, 0, 0 }, { 0, 0, 0, -1, 1 }, { 0, 0, 0, 0, 1 } });
00358
00359 /// <summary>
00360 /// A <see cref="ColourMatrix"/> that shifts every colour component by an amount corresponding to the
      inverted alpha value.  The alpha component is left intact.
00361 /// </summary>
00362           public static ColourMatrix InvertedAlphaShift = new ColourMatrix(new double[,] { { 1, 0, 0,
      -1, 1 }, { 0, 1, 0, -1, 1 }, { 0, 0, 1, -1, 1 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } });
00363
00364 /// <summary>
00365 /// Creates a <see cref="ColourMatrix"/> that turns every colour to which it is applied into the
      specified <paramref name="colour"/>.
00366 /// </summary>
00367 /// <param name="colour">The colour that will be produced by the <see cref="ColourMatrix"/>.</param>
00368 /// <param name="useAlpha">If this is <see langword="true"/>, all output pixels will have the same
      alpha value as the supplied <paramref name="colour"/>.  If this is false, the alpha value of the input
      pixels is preserved.</param>
00369 /// <returns>A <see cref="ColourMatrix"/> that turns every colour to which it is applied into the
      specified <paramref name="colour"/>.</returns>
00370           public static ColourMatrix ToColour(Colour colour, bool useAlpha = false)
00371           {
00372               if (!useAlpha)
00373               {
00374                   return new ColourMatrix(new double[,] { { 0, 0, 0, 0, colour.R }, { 0, 0, 0, 0,
      colour.G }, { 0, 0, 0, 0, colour.B }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } });
00375               }
00376               else
00377               {
00378                   return new ColourMatrix(new double[,] { { 0, 0, 0, 0, colour.R }, { 0, 0, 0, 0,
      colour.G }, { 0, 0, 0, 0, colour.B }, { 0, 0, 0, 0, colour.A }, { 0, 0, 0, 0, 1 } });
00379               }
00380           }
00381
00382 /// <summary>
00383 /// Creates a <see cref="ColourMatrix"/> that turns every colour to which it is applied into a shade
      of the specified <paramref name="colour"/>.  The brightness of the output colour depends on the
      luminance of the input colour.
00384 /// </summary>
00385 /// <param name="colour">The colour whose shades will be produced by the <see
      cref="ColourMatrix"/>.</param>
```

```
00386 /// <param name="useAlpha">If this is <see langword="true"/>, the transformation will also be applied
      to the alpha channel.  If this is false, the alpha value of the input pixels is preserved.</param>
00387 /// <returns>A <see cref="ColourMatrix"/> that turns every colour to which it is applied into a shade
      of the specified <paramref name="colour"/>.</returns>
00388         public static ColourMatrix LuminanceToColour(Colour colour, bool useAlpha = false)
00389         {
00390             if (!useAlpha)
00391             {
00392                 return new ColourMatrix(new double[,] { { 0.2126 * colour.R, 0.7152 * colour.R, 0.0722
      * colour.R, 0, 0 }, { 0.2126 * colour.G, 0.7152 * colour.G, 0.0722 * colour.G, 0, 0 }, { 0.2126 *
      colour.B, 0.7152 * colour.B, 0.0722 * colour.B, 0, 0 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 0, 1 } });
00393             }
00394             else
00395             {
00396                 return new ColourMatrix(new double[,] { { 0.2126 * colour.R, 0.7152 * colour.R, 0.0722
      * colour.R, 0, 0 }, { 0.2126 * colour.G, 0.7152 * colour.G, 0.0722 * colour.G, 0, 0 }, { 0.2126 *
      colour.B, 0.7152 * colour.B, 0.0722 * colour.B, 0, 0 }, { 0.2126 * colour.A, 0.7152 * colour.A, 0.0722
      * colour.A, 0, 0 }, { 0, 0, 0, 0, 1 } });
00397             }
00398         }
00399
00400 /// <summary>
00401 /// Creates a <see cref="ColourMatrix"/> that transforms the alpha value of the colour it is applied
      to into a value depending on the luminance of the input colour.
00402 /// </summary>
00403 /// <param name="preserveColour">If this is <see langword="true"/>, the values of the red, green and
      blue components of the input colour are preserved in the output colour.  If this is false, the output
      colour will always be black.</param>
00404 /// <returns>A <see cref="ColourMatrix"/> that transforms the alpha value of the colour it is applied
      to into a value depending on the luminance of the input colour.</returns>
00405         public static ColourMatrix LuminanceToAlpha(bool preserveColour = false)
00406         {
00407             if (!preserveColour)
00408             {
00409                 return new ColourMatrix(new double[,] { { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0,
      0, 0, 0 }, { 0.2126, 0.7152, 0.0722, 0, 0 }, { 0, 0, 0, 0, 1 } });
00410             }
00411             else
00412             {
00413                 return new ColourMatrix(new double[,] { { 1, 0, 0, 0, 0 }, { 0, 1, 0, 0, 0 }, { 0, 0,
      1, 0, 0 }, { 0.2126, 0.7152, 0.0722, 0, 0 }, { 0, 0, 0, 0, 1 } });
00414             }
00415         }
00416
00417 /// <summary>
00418 /// Creates a new <see cref="ColourMatrix"/> whose alpha coefficients are multiplied by the specified
      value.
00419 /// </summary>
00420 /// <param name="alpha">The value that will be used to multiply all the alpha coefficients of the <see
      cref="ColourMatrix"/>.</param>
00421 /// <returns>A new <see cref="ColourMatrix"/> whose alpha coefficients have been multiplied by the
      specified value.</returns>
00422         [Pure]
00423         public ColourMatrix WithAlpha(double alpha)
00424         {
00425             return new ColourMatrix(new double[,] { { R1, R2, R3, R4, R5 }, { G1, G2, G3, G4, G5 }, {
      B1, B2, B3, B4, B5 }, { A1 * alpha, A2 * alpha, A3 * alpha, A4 * alpha, A5 * alpha }, { 0, 0, 0, 0, 1
      } });
00426         }
00427
00428 /// <summary>
00429 /// Concatenates two matrices.  The resulting <see cref="ColourMatrix"/> is equivalent to first
      applying <paramref name="matrix2"/>, and then <paramref name="matrix1"/>.
00430 /// </summary>
00431 /// <param name="matrix1">The matrix that acts second.</param>
00432 /// <param name="matrix2">The matrix that acts first.</param>
00433 /// <returns>A <see cref="ColourMatrix"/> equivalent to first applying <paramref name="matrix2"/>, and
      then <paramref name="matrix1"/>.</returns>
00434         public static ColourMatrix operator *(ColourMatrix matrix1, ColourMatrix matrix2)
00435         {
00436             double[,] tbr = new double[5, 5];
00437
00438             for (int i = 0; i < 5; i++)
00439             {
00440                 for (int j = 0; j < 5; j++)
00441                 {
00442                     for (int k = 0; k < 5; k++)
00443                     {
00444                         tbr[i, j] += matrix1[i, k] * matrix2[k, j];
00445                     }
00446                 }
00447             }
00448
00449             return new ColourMatrix(tbr);
00450         }
00451
00452 /// <summary>
```

```
00453 /// Creates a new <see cref="ColourMatrix"/> with the specified coefficients.
00454 /// </summary>
00455 /// <param name="matrix">The coefficients of the <see cref="ColourMatrix"/>.</param>
00456         public ColourMatrix(double[,] matrix)
00457         {
00458             for (int i = 0; i < 4; i++)
00459             {
00460                 for (int j = 0; j < 5; j++)
00461                 {
00462                     this[i, j] = matrix[i, j];
00463                 }
00464             }
00465         }
00466 
00467 /// <summary>
00468 /// Applies the <see cref="ColourMatrix"/> to the specified <see cref="Colour"/>.
00469 /// </summary>
00470 /// <param name="colour">The <see cref="Colour"/> to which the <see cref="ColourMatrix"/> should be
      applied.</param>
00471 /// <returns>The result of applying the <see cref="ColourMatrix"/> to the specified colour.</returns>
00472         [Pure]
00473         public Colour Apply(Colour colour)
00474         {
00475             double[] col = new double[] { colour.R, colour.G, colour.B, colour.A, 1 };
00476 
00477             double[] tbr = new double[5];
00478 
00479             for (int i = 0; i < tbr.Length; i++)
00480             {
00481                 for (int j = 0; j < tbr.Length; j++)
00482                 {
00483                     tbr[i] += this[i, j] * col[j];
00484                 }
00485             }
00486 
00487             return Colour.FromRgba(tbr[0], tbr[1], tbr[2], tbr[3]);
00488         }
00489 
00490 /// <summary>
00491 /// Applies the <see cref="ColourMatrix"/> to the specified colour, represented as four bytes, and
      stores the resulting colour in the
00492 /// same variables as the original RGBA values.
00493 /// </summary>
00494 /// <param name="R">The R component of the input colour.  After this method returns, this will contain
      the R component of the output colour.</param>
00495 /// <param name="G">The G component of the input colour.  After this method returns, this will contain
      the G component of the output colour.</param>
00496 /// <param name="B">The B component of the input colour.  After this method returns, this will contain
      the B component of the output colour.</param>
00497 /// <param name="A">The A component of the input colour.  After this method returns, this will contain
      the A component of the output colour.</param>
00498         public void Apply(ref byte R, ref byte G, ref byte B, ref byte A)
00499         {
00500             byte r = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.R1 + G * this.R2 + B *
      this.R3 + A * this.R4 + this.R5 * 255)));
00501             byte g = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.G1 + G * this.G2 + B *
      this.G3 + A * this.G4 + this.G5 * 255)));
00502             byte b = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.B1 + G * this.B2 + B *
      this.B3 + A * this.B4 + this.B5 * 255)));
00503             byte a = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.A1 + G * this.A2 + B *
      this.A3 + A * this.A4 + this.A5 * 255)));
00504 
00505             R = r;
00506             G = g;
00507             B = b;
00508             A = a;
00509         }
00510 
00511 /// <summary>
00512 /// Applies the <see cref="ColourMatrix"/> to the specified colour, represented as three bytes, and
      stores the resulting colour in the
00513 /// same variables as the original RGB values.
00514 /// </summary>
00515 /// <param name="R">The R component of the input colour.  After this method returns, this will contain
      the R component of the output colour.</param>
00516 /// <param name="G">The G component of the input colour.  After this method returns, this will contain
      the G component of the output colour.</param>
00517 /// <param name="B">The B component of the input colour.  After this method returns, this will contain
      the B component of the output colour.</param>
00518         public void Apply(ref byte R, ref byte G, ref byte B)
00519         {
00520             byte r = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.R1 + G * this.R2 + B *
      this.R3 + 255 * this.R4 + this.R5 * 255)));
00521             byte g = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.G1 + G * this.G2 + B *
      this.G3 + 255 * this.G4 + this.G5 * 255)));
00522             byte b = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.B1 + G * this.B2 + B *
      this.B3 + 255 * this.B4 + this.B5 * 255)));
```

```
00523
00524             R = r;
00525             G = g;
00526             B = b;
00527         }
00528
00529 /// <summary>
00530 /// Applies the <see cref="ColourMatrix"/> to the specified colour, represented as four bytes, and
      stores the resulting colour in the
00531 /// specified output bytes.
00532 /// </summary>
00533 /// <param name="R">The R component of the input colour.</param>
00534 /// <param name="G">The G component of the input colour.</param>
00535 /// <param name="B">The B component of the input colour.</param>
00536 /// <param name="A">The A component of the input colour.</param>
00537 /// <param name="r">After this method returns, this will contain the R component of the output
      colour.</param>
00538 /// <param name="g">After this method returns, this will contain the G component of the output
      colour.</param>
00539 /// <param name="b">After this method returns, this will contain the B component of the output
      colour.</param>
00540 /// <param name="a">After this method returns, this will contain the A component of the output
      colour.</param>
00541         public void Apply(byte R, byte G, byte B, byte A, out byte r, out byte g, out byte b, out byte
      a)
00542         {
00543             r = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.R1 + G * this.R2 + B * this.R3 + A
      * this.R4 + this.R5 * 255)));
00544             g = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.G1 + G * this.G2 + B * this.G3 + A
      * this.G4 + this.G5 * 255)));
00545             b = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.B1 + G * this.B2 + B * this.B3 + A
      * this.B4 + this.B5 * 255)));
00546             a = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.A1 + G * this.A2 + B * this.A3 + A
      * this.A4 + this.A5 * 255)));
00547         }
00548
00549 /// <summary>
00550 /// Applies the <see cref="ColourMatrix"/> to the specified colour, represented as three bytes, and
      stores the resulting colour in the
00551 /// specified output bytes.
00552 /// </summary>
00553 /// <param name="R">The R component of the input colour.</param>
00554 /// <param name="G">The G component of the input colour.</param>
00555 /// <param name="B">The B component of the input colour.</param>
00556 /// <param name="r">After this method returns, this will contain the R component of the output
      colour.</param>
00557 /// <param name="g">After this method returns, this will contain the G component of the output
      colour.</param>
00558 /// <param name="b">After this method returns, this will contain the B component of the output
      colour.</param>
00559         public void Apply(byte R, byte G, byte B, out byte r, out byte g, out byte b)
00560         {
00561             r = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.R1 + G * this.R2 + B * this.R3 +
      255 * this.R4 + this.R5 * 255)));
00562             g = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.G1 + G * this.G2 + B * this.G3 +
      255 * this.G4 + this.G5 * 255)));
00563             b = (byte)Math.Min(255, Math.Max(0, Math.Round(R * this.B1 + G * this.B2 + B * this.B3 +
      255 * this.B4 + this.B5 * 255)));
00564         }
00565     }
00566
00567 /// <summary>
00568 /// Represents a filter that applies a <see cref="Filters.ColourMatrix"/> to the colours of the image.
00569 /// </summary>
00570     public class ColourMatrixFilter :  ILocationInvariantFilter
00571     {
00572 /// <summary>
00573 /// The <see cref="Filters.ColourMatrix"/> that is applied by this filter.
00574 /// </summary>
00575         public ColourMatrix ColourMatrix { get; }
00576
00577 /// <inheritdoc/>
00578         public Point TopLeftMargin { get; } = new Point();
00579
00580 /// <inheritdoc/>
00581         public Point BottomRightMargin { get; } = new Point();
00582
00583 /// <summary>
00584 /// Creates a new <see cref="ColourMatrixFilter"/> with the specified <see
      cref="Filters.ColourMatrix"/>.
00585 /// </summary>
00586 /// <param name="colorMatrix">The <see cref="Filters.ColourMatrix"/> that will be applied by the
      filter.</param>
00587         public ColourMatrixFilter(ColourMatrix colorMatrix)
00588         {
00589             this.ColourMatrix = colorMatrix;
00590         }
```

```
00591
00592 /// <inheritdoc/>
00593         [Pure]
00594         public RasterImage Filter(RasterImage image, double scale)
00595         {
00596             IntPtr tbrData = System.Runtime.InteropServices.Marshal.AllocHGlobal(image.Width *
      image.Height * (image.HasAlpha ?  4 :  3));
00597             GC.AddMemoryPressure(image.Width * image.Height * (image.HasAlpha ?  4 :  3));
00598
00599             int width = image.Width;
00600             int height = image.Height;
00601
00602             int pixelSize = image.HasAlpha ?  4 :  3;
00603             int stride = image.Width * pixelSize;
00604
00605             int threads = Math.Min(8, Environment.ProcessorCount);
00606
00607             unsafe
00608             {
00609                 byte* input = (byte*)image.ImageDataAddress;
00610                 byte* output = (byte*)tbrData;
00611
00612                 Action<int> yLoop;
00613
00614                 if (image.HasAlpha)
00615                 {
00616                     yLoop = (y) =>
00617                     {
00618                         for (int x = 0; x < width; x++)
00619                         {
00620                             this.ColourMatrix.Apply(input[y * stride + x * 4], input[y * stride + x *
      4 + 1], input[y * stride + x * 4 + 2], input[y * stride + x * 4 + 3], out output[y * stride + x * 4],
      out output[y * stride + x * 4 + 1], out output[y * stride + x * 4 + 2], out output[y * stride + x * 4
      + 3]);
00621                         }
00622                     };
00623                 }
00624                 else
00625                 {
00626                     yLoop = (y) =>
00627                     {
00628                         for (int x = 0; x < width; x++)
00629                         {
00630                             this.ColourMatrix.Apply(input[y * stride + x * 4], input[y * stride + x *
      4 + 1], input[y * stride + x * 4 + 2], out output[y * stride + x * 4], out output[y * stride + x * 4 +
      1], out output[y * stride + x * 4 + 2]);
00631                         }
00632                     };
00633                 }
00634
00635                 if (threads == 1)
00636                 {
00637                     for (int y = 0; y < height; y++)
00638                     {
00639                         yLoop(y);
00640                     }
00641                 }
00642                 else
00643                 {
00644                     ParallelOptions options = new ParallelOptions() { MaxDegreeOfParallelism = threads
      };
00645
00646                     Parallel.For(0, height, options, yLoop);
00647                 }
00648
00649             }
00650
00651             DisposableIntPtr disp = new DisposableIntPtr(tbrData);
00652
00653             return new RasterImage(ref disp, width, height, image.HasAlpha, image.Interpolate);
00654         }
00655     }
00656 }
```

8.74 CompositeFilter.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
```

```
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System.Collections.Generic;
00019 using System.Collections.Immutable;
00020 using System.Diagnostics.Contracts;
00021
00022 namespace VectSharp.Filters
00023 {
00024 /// <summary>
00025 /// Represents a filter that corresponds to applying multiple <see cref="ILocationInvariantFilter"/>s
      one after the other.
00026 /// </summary>
00027     public class CompositeLocationInvariantFilter :  ILocationInvariantFilter
00028     {
00029 /// <inheritdoc/>
00030         public Point TopLeftMargin { get; }
00031
00032 /// <inheritdoc/>
00033         public Point BottomRightMargin { get; }
00034
00035 /// <summary>
00036 /// The filters that are applied by this filter.
00037 /// </summary>
00038         public ImmutableList<ILocationInvariantFilter> Filters { get; }
00039
00040 /// <summary>
00041 /// Creates a new <see cref="CompositeLocationInvariantFilter"/> with the specified filters.
00042 /// </summary>
00043 /// <param name="filters">The filters that will be applied by the new filter.</param>
00044         public CompositeLocationInvariantFilter(IEnumerable<ILocationInvariantFilter> filters)
00045         {
00046             IEnumerable<ILocationInvariantFilter> flattenedFilters = FlattenFilters(filters);
00047             this.Filters = ImmutableList.CreateRange(flattenedFilters);
00048
00049             bool initialised = false;
00050
00051             foreach (IFilter filter in flattenedFilters)
00052             {
00053                 if (!initialised)
00054                 {
00055                     this.TopLeftMargin = filter.TopLeftMargin;
00056                     this.BottomRightMargin = filter.BottomRightMargin;
00057                     initialised = true;
00058                 }
00059                 else
00060                 {
00061                     this.TopLeftMargin = Point.Max(this.TopLeftMargin, filter.TopLeftMargin);
00062                     this.BottomRightMargin = Point.Max(this.BottomRightMargin,
      filter.BottomRightMargin);
00063                 }
00064             }
00065         }
00066
00067 /// <summary>
00068 /// Creates a new <see cref="CompositeLocationInvariantFilter"/> with the specified filters.
00069 /// </summary>
00070 /// <param name="filters">The filters that will be applied by the new filter.</param>
00071         public CompositeLocationInvariantFilter(params ILocationInvariantFilter[] filters) :
      this((IEnumerable<ILocationInvariantFilter>)filters) {  }
00072
00073         private IEnumerable<ILocationInvariantFilter>
      FlattenFilters(IEnumerable<ILocationInvariantFilter> filters)
00074         {
00075             foreach (ILocationInvariantFilter filter in filters)
00076             {
00077                 if (filter is CompositeLocationInvariantFilter composite)
00078                 {
00079                     foreach (ILocationInvariantFilter filter2 in FlattenFilters(composite.Filters))
00080                     {
00081                         yield return filter2;
00082                     }
00083                 }
00084                 else
00085                 {
00086                     yield return filter;
00087                 }
00088             }
00089         }
00090
```

```
00091 /// <inheritdoc/>
00092         [Pure]
00093         public RasterImage Filter(RasterImage image, double scale)
00094         {
00095             RasterImage currImage = image;
00096 
00097             foreach (ILocationInvariantFilter filter in this.Filters)
00098             {
00099                 RasterImage prevImage = currImage;
00100                 currImage = filter.Filter(prevImage, scale);
00101 
00102                 if (prevImage != image)
00103                 {
00104                     prevImage.Dispose();
00105                 }
00106             }
00107 
00108             return currImage;
00109         }
00110     }
00111 }
```

8.75 ConvolutionFilter.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
00019 using System.Diagnostics.Contracts;
00020 using System.Threading.Tasks;
00021 
00022 namespace VectSharp.Filters
00023 {
00024 /// <summary>
00025 /// Represents a filter that applies a matrix convolution to the image.
00026 /// </summary>
00027     public class ConvolutionFilter :  ILocationInvariantFilter
00028     {
00029 /// <inheritdoc/>
00030         public Point TopLeftMargin { get; protected set; }
00031 /// <inheritdoc/>
00032         public Point BottomRightMargin { get; protected set; }
00033 
00034 /// <summary>
00035 /// The kernel of the <see cref="ConvolutionFilter"/>.  The dimensions of this matrix should all be
      odd numbers.  The larger the kernel, the worse the performance.
00036 /// </summary>
00037         public virtual double[,] Kernel { get; protected set; }
00038 
00039 /// <summary>
00040 /// The normalisation value that is applies to the kernel.
00041 /// </summary>
00042         public virtual double Normalisation { get; protected set; } = 1;
00043 
00044 /// <summary>
00045 /// The bias value that is added to every colour component when the filter is applied.
00046 /// </summary>
00047         public virtual double Bias { get; protected set; } = 0;
00048 
00049 /// <summary>
00050 /// The scale relating the size of the kernel to graphics units.
00051 /// </summary>
00052         public virtual double Scale { get; protected set; }
00053 
00054 /// <summary>
00055 /// If this is <see langword="true"/>, the alpha value of the input pixels is preserved.  Otherwise,
      the alpha channel is subject to the same convolution process as the other colour components.
00056 /// </summary>
00057         public virtual bool PreserveAlpha { get; protected set; } = true;
```

```
00058
00059 /// <summary>
00060 /// Creates a new <see cref="ConvolutionFilter"/> with the specified parameters.
00061 /// </summary>
00062 /// <param name="kernel">The kernel of the <see cref="ConvolutionFilter"/>.  The dimensions of this
      matrix should all be odd numbers.  The larger the kernel, the worse the performance.</param>
00063 /// <param name="scale">The scale relating the size of the kernel to graphics units.</param>
00064 /// <param name="preserveAlpha">If this is <see langword="true"/>, the alpha value of the input pixels
      is preserved.  Otherwise, the alpha channel is subject to the same convolution process as the other
      colour components.</param>
00065 /// <param name="normalisation">The normalisation value that is applies to the kernel.</param>
00066 /// <param name="bias">The bias value that is added to every colour component when the filter is
      applied.</param>
00067 /// <exception cref="ArgumentException">This exception is thrown when the kernel dimensions are not
      odd numbers.</exception>
00068         public ConvolutionFilter(double[,] kernel, double scale, bool preserveAlpha = true, double
      normalisation = 1, double bias = 0)
00069         {
00070             if (kernel.GetLength(0) % 2 != 1 || kernel.GetLength(1) % 2 != 1)
00071             {
00072                 throw new ArgumentException("The kernel must have an odd number of rows and columns!",
      nameof(kernel));
00073             }
00074
00075             this.Kernel = kernel;
00076
00077             int kernelWidth = (this.Kernel.GetLength(0) - 1) / 2;
00078             int kernelHeight = (this.Kernel.GetLength(1) - 1) / 2;
00079
00080             this.TopLeftMargin = new Point(kernelWidth * scale, kernelHeight * scale);
00081             this.BottomRightMargin = new Point(kernelWidth * scale, kernelHeight * scale);
00082             this.Scale = scale;
00083             this.PreserveAlpha = preserveAlpha;
00084             this.Normalisation = normalisation;
00085             this.Bias = bias;
00086         }
00087
00088 /// <inheritdoc/>
00089         [Pure]
00090         public virtual RasterImage Filter(RasterImage image, double scale)
00091         {
00092             IntPtr tbrData = System.Runtime.InteropServices.Marshal.AllocHGlobal(image.Width *
      image.Height * (image.HasAlpha ?  4 :  3));
00093
00094             int width = image.Width;
00095             int height = image.Height;
00096
00097             int kernelWidth = (this.Kernel.GetLength(0) - 1) / 2;
00098             int kernelHeight = (this.Kernel.GetLength(1) - 1) / 2;
00099
00100             int actualKernelWidth = (int)Math.Round(kernelWidth * scale * this.Scale);
00101             int actualKernelHeight = (int)Math.Round(kernelHeight * scale * this.Scale);
00102
00103             actualKernelWidth = Math.Max(actualKernelWidth, 1);
00104             actualKernelHeight = Math.Max(actualKernelHeight, 1);
00105
00106             int[] kernelX = new int[actualKernelWidth * 2 + 1];
00107
00108             for (int x = 0; x < actualKernelWidth * 2 + 1; x++)
00109             {
00110                 kernelX[x] = (int)Math.Round((double)(x - actualKernelWidth) / actualKernelWidth *
      kernelWidth + kernelWidth);
00111             }
00112
00113             int[] kernelY = new int[actualKernelHeight * 2 + 1];
00114
00115             for (int y = 0; y < actualKernelHeight * 2 + 1; y++)
00116             {
00117                 kernelY[y] = (int)Math.Round((double)(y - actualKernelHeight) / actualKernelHeight *
      kernelHeight + kernelHeight);
00118             }
00119
00120             int[] countsX = new int[2 * kernelWidth + 1];
00121
00122             for (int i = 0; i < 2 * actualKernelWidth + 1; i++)
00123             {
00124                 countsX[kernelX[i]]++;
00125             }
00126
00127             int[] countsY = new int[2 * kernelHeight + 1];
00128
00129             for (int i = 0; i < 2 * actualKernelHeight + 1; i++)
00130             {
00131                 countsY[kernelY[i]]++;
00132             }
00133
00134
```

```
00135             double[] weightsX = new double[2 * actualKernelWidth + 1];
00136
00137             for (int i = 0; i < 2 * actualKernelWidth + 1; i++)
00138             {
00139                 weightsX[i] = 1.0 / countsX[kernelX[i]];
00140             }
00141
00142             double[] weightsY = new double[2 * actualKernelHeight + 1];
00143
00144             for (int i = 0; i < 2 * actualKernelHeight + 1; i++)
00145             {
00146                 weightsY[i] = 1.0 / countsY[kernelY[i]];
00147             }
00148
00149             kernelWidth = actualKernelWidth;
00150             kernelHeight = actualKernelHeight;
00151
00152             double normalisation = this.Normalisation;
00153
00154             if (double.IsNaN(normalisation) || normalisation == 0)
00155             {
00156                 normalisation = 1;
00157             }
00158
00159             double totalWeight = 0;
00160             for (int i = 0; i < kernelWidth * 2 + 1; i++)
00161             {
00162                 for (int j = 0; j < kernelHeight * 2 + 1; j++)
00163                 {
00164                     totalWeight += Kernel[kernelX[i], kernelY[j]] * weightsX[i] * weightsY[j];
00165                 }
00166             }
00167
00168             if (Math.Abs(totalWeight) < 1e-5)
00169             {
00170                 totalWeight = 1;
00171             }
00172
00173
00174             double bias = this.Bias * 255;
00175             int pixelSize = image.HasAlpha ?  4 :  3;
00176             int stride = image.Width * pixelSize;
00177
00178             int threads = Math.Min(8, Environment.ProcessorCount);
00179
00180             unsafe
00181             {
00182                 byte* input = (byte*)image.ImageDataAddress;
00183                 byte* output = (byte*)tbrData;
00184
00185                 Action<int> yLoop;
00186
00187                 if (image.HasAlpha && !PreserveAlpha)
00188                 {
00189                     yLoop = (y) =>
00190                     {
00191                         for (int x = 0; x < width; x++)
00192                         {
00193                             double R = 0;
00194                             double G = 0;
00195                             double B = 0;
00196
00197                             double weight = 0;
00198
00199                             for (int targetX = 0; targetX <= kernelWidth * 2; targetX++)
00200                             {
00201                                 for (int targetY = 0; targetY <= kernelHeight * 2; targetY++)
00202                                 {
00203                                     int tX = Math.Min(Math.Max(0, x + targetX - kernelWidth), width -
      1);
00204                                     int tY = Math.Min(Math.Max(0, y + targetY - kernelHeight), height
      - 1);
00205
00206                                     double a = input[tY * stride + tX * 4 + 3] / 255.0 *
      weightsX[targetX] * weightsY[targetY];
00207
00208                                     int projectedX = kernelX[targetX];
00209                                     int projectedY = kernelY[targetY];
00210
00211                                     weight += Kernel[projectedX, projectedY] * a;
00212
00213                                     R += Kernel[projectedX, projectedY] * input[tY * stride + tX * 4]
      * a;
00214                                     G += Kernel[projectedX, projectedY] * input[tY * stride + tX * 4 +
      1] * a;
00215                                     B += Kernel[projectedX, projectedY] * input[tY * stride + tX * 4 +
      2] * a;
```

```
00216                                 }
00217                             }
00218 
00219                             if (weight != 0)
00220                             {
00221                                 output[y * stride + x * 4] = (byte)Math.Min(255, Math.Max(0, R /
      (normalisation * weight) + bias));
00222                                 output[y * stride + x * 4 + 1] = (byte)Math.Min(255, Math.Max(0, G /
      (normalisation * weight) + bias));
00223                                 output[y * stride + x * 4 + 2] = (byte)Math.Min(255, Math.Max(0, B /
      (normalisation * weight) + bias));
00224                                 output[y * stride + x * 4 + 3] = (byte)Math.Min(255, Math.Max(0,
      (weight / (normalisation * totalWeight) * 255 + bias)));
00225                             }
00226                             else
00227                             {
00228                                 output[y * stride + x * 4] = 0;
00229                                 output[y * stride + x * 4 + 1] = 0;
00230                                 output[y * stride + x * 4 + 2] = 0;
00231                                 output[y * stride + x * 4 + 3] = 0;
00232                             }
00233                         }
00234                     };
00235                 }
00236                 else if (image.HasAlpha && PreserveAlpha)
00237                 {
00238                     yLoop = (y) =>
00239                     {
00240                         for (int x = 0; x < width; x++)
00241                         {
00242                             double R = 0;
00243                             double G = 0;
00244                             double B = 0;
00245 
00246                             for (int targetX = 0; targetX <= kernelWidth * 2; targetX++)
00247                             {
00248                                 for (int targetY = 0; targetY <= kernelHeight * 2; targetY++)
00249                                 {
00250                                     int tX = Math.Min(Math.Max(0, x + targetX - kernelWidth), width -
      1);
00251                                     int tY = Math.Min(Math.Max(0, y + targetY - kernelHeight), height
      - 1);
00252 
00253                                     int projectedX = kernelX[targetX];
00254                                     int projectedY = kernelY[targetY];
00255 
00256                                     R += Kernel[projectedX, projectedY] * input[tY * stride + tX * 4]
      * weightsX[targetX] * weightsY[targetY];
00257                                     G += Kernel[projectedX, projectedY] * input[tY * stride + tX * 4 +
      1] * weightsX[targetX] * weightsY[targetY];
00258                                     B += Kernel[projectedX, projectedY] * input[tY * stride + tX * 4 +
      2] * weightsX[targetX] * weightsY[targetY];
00259                                 }
00260                             }
00261 
00262                             output[y * stride + x * 4] = (byte)Math.Min(255, Math.Max(0, R /
      (normalisation * totalWeight) + bias));
00263                             output[y * stride + x * 4 + 1] = (byte)Math.Min(255, Math.Max(0, G /
      (normalisation * totalWeight) + bias));
00264                             output[y * stride + x * 4 + 2] = (byte)Math.Min(255, Math.Max(0, B /
      (normalisation * totalWeight) + bias));
00265                             output[y * stride + x * 4 + 3] = input[y * stride + x * 4 + 3];
00266                         }
00267                     };
00268                 }
00269                 else
00270                 {
00271                     yLoop = (y) =>
00272                     {
00273                         for (int x = 0; x < width; x++)
00274                         {
00275                             double R = 0;
00276                             double G = 0;
00277                             double B = 0;
00278 
00279                             for (int targetX = 0; targetX <= kernelWidth * 2; targetX++)
00280                             {
00281                                 for (int targetY = 0; targetY <= kernelHeight * 2; targetY++)
00282                                 {
00283                                     int tX = Math.Min(Math.Max(0, x + targetX - kernelWidth), width -
      1);
00284                                     int tY = Math.Min(Math.Max(0, y + targetY - kernelHeight), height
      - 1);
00285 
00286                                     int projectedX = kernelX[targetX];
00287                                     int projectedY = kernelY[targetY];
00288 
```

```
00289                                   R += Kernel[projectedX, projectedY] * input[tY * stride + tX * 3]
      * weightsX[targetX] * weightsY[targetY];
00290                                   G += Kernel[projectedX, projectedY] * input[tY * stride + tX * 3 +
      1] * weightsX[targetX] * weightsY[targetY];
00291                                   B += Kernel[projectedX, projectedY] * input[tY * stride + tX * 3 +
      2] * weightsX[targetX] * weightsY[targetY];
00292                               }
00293                           }
00294
00295                           output[y * stride + x * 3] = (byte)Math.Min(255, Math.Max(0, R /
      (normalisation * totalWeight) + bias));
00296                           output[y * stride + x * 3 + 1] = (byte)Math.Min(255, Math.Max(0, G /
      (normalisation * totalWeight) + bias));
00297                           output[y * stride + x * 3 + 2] = (byte)Math.Min(255, Math.Max(0, B /
      (normalisation * totalWeight) + bias));
00298                       }
00299                   };
00300               }
00301
00302               if (threads == 1)
00303               {
00304                   for (int y = 0; y < height; y++)
00305                   {
00306                       yLoop(y);
00307                   }
00308               }
00309               else
00310               {
00311                   ParallelOptions options = new ParallelOptions() { MaxDegreeOfParallelism = threads
      };
00312
00313                   Parallel.For(0, height, options, yLoop);
00314               }
00315           }
00316
00317           return new RasterImage(tbrData, image.Width, image.Height, image.HasAlpha,
      image.Interpolate);
00318       }
00319   }
00320 }
```

8.76 Filters.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Diagnostics.Contracts;
00020
00021 namespace VectSharp.Filters
00022 {
00023 /// <summary>
00024 /// Represents a filter.  Do not implement this interface directly; instead, implement <see
      cref="ILocationInvariantFilter"/> or <see cref="IFilterWithLocation"/>.
00025 /// </summary>
00026     public interface IFilter
00027     {
00028 /// <summary>
00029 /// Determines how much the area of the filter's subject should be expanded on the top-left to
      accommodate the results of the filter.
00030 /// </summary>
00031         Point TopLeftMargin { get; }
00032
00033 /// <summary>
00034 /// Determines how much the area of the filter's subject should be expanded on the bottom-right to
      accommodate the results of the filter.
00035 /// </summary>
00036         Point BottomRightMargin { get; }
00037     }
```

```
00038
00039 /// <summary>
00040 /// Represents a filter that can be applied to an image regardless of its location on the graphics
      surface.
00041 /// </summary>
00042     public interface ILocationInvariantFilter :  IFilter
00043     {
00044 /// <summary>
00045 /// Applies the filter to a <see cref="RasterImage"/>.
00046 /// </summary>
00047 /// <param name="image">The <see cref="RasterImage"/> to which the filter will be applied.</param>
00048 /// <param name="scale">The scale of the image with respect to the filter.</param>
00049 /// <returns>A new <see cref="RasterImage"/> containing the filtered image.  The source <paramref
      name="image"/> is left unaltered.</returns>
00050         [Pure]
00051         RasterImage Filter(RasterImage image, double scale);
00052     }
00053
00054 /// <summary>
00055 /// Represents a filter whose results depend on the position of the subject image on the graphics
      surface.
00056 /// </summary>
00057     public interface IFilterWithLocation :  IFilter
00058     {
00059 /// <summary>
00060 /// Applies the filter to a <see cref="RasterImage"/>.
00061 /// </summary>
00062 /// <param name="image">The <see cref="RasterImage"/> to which the filter will be applied.</param>
00063 /// <param name="bounds">The region on the graphics surface where the image will be drawn.</param>
00064 /// <param name="scale">The scale of the image with respect to the filter.</param>
00065 /// <returns>A new <see cref="RasterImage"/> containing the filtered image.  The source <paramref
      name="image"/> is left unaltered.</returns>
00066         [Pure]
00067         RasterImage Filter(RasterImage image, Rectangle bounds, double scale);
00068     }
00069
00070 /// <summary>
00071 /// Represents a filter with a parameter that needs to be rasterised at the same resolution as the
      subject image prior to applying the filter.
00072 /// The <see cref="FilterWithRasterisableParameter"/> abstract class provides a default implementation
      of this interface.
00073 /// </summary>
00074     public interface IFilterWithRasterisableParameter
00075     {
00076 /// <summary>
00077 /// Rasterises the filter's parameter at the specified scale, using the specified rasterisation
      method.
00078 /// </summary>
00079 /// <param name="rasterisationMethod">The method used to rasterise the image.  The first argument of
      this method is the <see cref="Graphics"/> to be rasterised,
00080 /// the second is a <see cref="Rectangle"/> representing the region to rasterise, the third is a <see
      cref="double"/> representing the scale, and the third is
00081 /// a boolean value indicating whether the resulting <see cref="RasterImage"/> should be
      interpolated.</param>
00082 /// <param name="scale">The scale factor at which the parameter is rasterised.</param>
00083         void RasteriseParameter(Func<Graphics, Rectangle, double, bool, RasterImage>
      rasterisationMethod, double scale);
00084     }
00085
00086 /// <summary>
00087 /// Represents a filter with a parameter that needs to be rasterised at the same resolution as the
      subject image prior to applying the filter.
00088 /// </summary>
00089     public abstract class FilterWithRasterisableParameter :  IFilterWithRasterisableParameter,
      IDisposable
00090     {
00091 /// <summary>
00092 /// The parameter that needs to be rasterised.
00093 /// </summary>
00094         protected virtual Graphics RasterisableParameter { get; }
00095
00096 /// <summary>
00097 /// The result of the last rasterisation of the <see cref="RasterisableParameter"/>.
00098 /// </summary>
00099         protected RasterImage cachedRasterisation;
00100         private bool disposedValue;
00101
00102 /// <summary>
00103 /// Get a rasterised version of the <see cref="RasterisableParameter"/> at the specified scale.  If
      this has already been computed, the cached result is returned.
00104 /// Otherwise, the image is rasterised using the default rasterisation method.
00105 /// </summary>
00106 /// <param name="scale">The scale at which the <see cref="RasterisableParameter"/> will be
      rasterised.</param>
00107 /// <returns>A rasterised version of the <see cref="RasterisableParameter"/> at the specified
      scale.</returns>
00108         protected virtual RasterImage GetCachedRasterisation(double scale)
```

```
00109         {
00110             if (this.cachedRasterisation == null || CachedResolution != scale)
00111             {
00112                 this.RasteriseParameter(scale);
00113             }
00114 
00115             return this.cachedRasterisation;
00116         }
00117 
00118 /// <summary>
00119 /// The bounds of the last cached rendering of the <see cref="RasterisableParameter"/> on the graphics surface.
00120 /// </summary>
00121         protected virtual Rectangle CachedBounds { get; set; }
00122 
00123 /// <summary>
00124 /// The resolution using which the cached rendering of the <see cref="RasterisableParameter"/> has been computed.
00125 /// </summary>
00126         protected virtual double CachedResolution { get; set; } = double.NaN;
00127 
00128 /// <summary>
00129 /// Create a new <see cref="FilterWithRasterisableParameter"/> with the specified parameter.
00130 /// </summary>
00131 /// <param name="rasterisableParameter">The parameter that needs to be rasterised at the same resolution as the subject image prior to applying the filter.</param>
00132         protected FilterWithRasterisableParameter(Graphics rasterisableParameter)
00133         {
00134             this.RasterisableParameter = rasterisableParameter;
00135         }
00136 
00137 /// <inheritdoc />
00138         public virtual void RasteriseParameter(Func<Graphics, Rectangle, double, bool, RasterImage> rasterisationMethod, double scale)
00139         {
00140             Rectangle bounds = RasterisableParameter.GetBounds();
00141 
00142             this.cachedRasterisation?.Dispose();
00143 
00144             this.cachedRasterisation = rasterisationMethod(this.RasterisableParameter, bounds, scale, true);
00145             this.CachedResolution = scale;
00146             this.CachedBounds = bounds;
00147         }
00148 
00149 /// <summary>
00150 /// Rasterises the filter's parameter at the specified scale, using the default rasterisation method.
00151 /// </summary>
00152 /// <param name="scale">The scale factor at which the parameter is rasterised.</param>
00153 /// <exception cref="NotImplementedException">This exception is thrown when there is no default rasterisation method.  This occurs because neither the VectSharp.Raster
00154 /// assembly, nor the VectSharp.Raster.ImageSharp assembly have been loaded, and no custom implementation of <see cref="Graphics.RasterisationMethod"/> has been provided.</exception>
00155         protected virtual void RasteriseParameter(double scale)
00156         {
00157             Rectangle bounds = this.RasterisableParameter.GetBounds();
00158 
00159             this.cachedRasterisation?.Dispose();
00160 
00161             if (this.RasterisableParameter.TryRasterise(bounds, scale, true, out RasterImage raster))
00162             {
00163                 this.cachedRasterisation = raster;
00164                 this.CachedResolution = scale;
00165                 this.CachedBounds = bounds;
00166             }
00167             else
00168             {
00169                 throw new NotImplementedException(@"The filter could not be rasterised!  You can avoid this error by doing one of the following:
00170 • Add a reference to VectSharp.Raster or VectSharp.Raster.ImageSharp (you may also need to add a using directive somewhere to force the assembly to be loaded).
00171 • Provide your own implementation of Graphics.RasterisationMethod.");
00172             }
00173         }
00174 
00175 /// <inheritdoc/>
00176         protected virtual void Dispose(bool disposing)
00177         {
00178             if (!disposedValue)
00179             {
00180                 if (disposing)
00181                 {
00182                     this.cachedRasterisation?.Dispose();
00183                 }
00184 
00185                 disposedValue = true;
00186             }
```

```
00187         }
00188
00189 /// <inheritdoc/>
00190         public void Dispose()
00191         {
00192             Dispose(disposing:  true);
00193             GC.SuppressFinalize(this);
00194         }
00195     }
00196
00197
00198 }
```

8.77 GaussianBlurFilter.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Diagnostics.Contracts;
00020 using System.Threading.Tasks;
00021
00022 namespace VectSharp.Filters
00023 {
00024 /// <summary>
00025 /// Represents a filter that applies a Gaussian blur effect.
00026 /// </summary>
00027     public class GaussianBlurFilter :  ILocationInvariantFilter
00028     {
00029 /// <summary>
00030 /// The standard deviation of the Gaussian blur.
00031 /// </summary>
00032         public double StandardDeviation { get; }
00033
00034 /// <inheritdoc/>
00035         public Point TopLeftMargin { get; }
00036
00037 /// <inheritdoc/>
00038         public Point BottomRightMargin { get; }
00039
00040 /// <summary>
00041 /// Creates a new <see cref="GaussianBlurFilter"/> with the specified standard deviation.
00042 /// </summary>
00043 /// <param name="standardDeviation">The standard deviation of the Gaussian blur.</param>
00044         public GaussianBlurFilter(double standardDeviation)
00045         {
00046             this.StandardDeviation = standardDeviation;
00047             this.TopLeftMargin = new Point(standardDeviation * 3, standardDeviation * 3);
00048             this.BottomRightMargin = new Point(standardDeviation * 3, standardDeviation * 3);
00049         }
00050
00051 /// <inheritdoc/>
00052         [Pure]
00053         public RasterImage Filter(RasterImage image, double scale)
00054         {
00055             return FilterSRGB(image, scale);
00056         }
00057
00058         private RasterImage FilterSRGB(RasterImage image, double scale)
00059         {
00060             if (this.StandardDeviation * scale > 2)
00061             {
00062                 int[] boxes = BoxesForGauss(this.StandardDeviation * scale, 3);
00063
00064                 for (int i = 0; i < boxes.Length; i++)
00065                 {
00066                     BoxBlurFilter box = new BoxBlurFilter((boxes[i] - 1) / 2);
00067
00068                     image = box.Filter(image, 1);
```

```
00069                 }
00070             }
00071             else
00072             {
00073                 image = FilterSRGBExact(image, this.StandardDeviation * scale);
00074             }
00075 
00076             return image;
00077         }
00078 
00079         // Adapted from http://blog.ivank.net/fastest-gaussian-blur.html
00080         private static int[] BoxesForGauss(double standardDeviation, int count)
00081         {
00082             double idealWeight = Math.Sqrt((12 * standardDeviation * standardDeviation / count) + 1);
00083 
00084             int lowerWeight = (int)Math.Floor(idealWeight);
00085 
00086             if (lowerWeight % 2 == 0)
00087             {
00088                 lowerWeight--;
00089             }
00090 
00091             int upperWeight = lowerWeight + 2;
00092 
00093             double idealSize = (12 * standardDeviation * standardDeviation - count * lowerWeight *
    lowerWeight - 4 * count * lowerWeight - 3 * count) / (-4 * lowerWeight - 4);
00094 
00095             int m = (int)Math.Round(idealSize);
00096 
00097             int[] sizes = new int[count];
00098 
00099             for (int i = 0; i < count; i++)
00100             {
00101                 sizes[i] = i < m ?  lowerWeight :  upperWeight;
00102             }
00103 
00104             return sizes;
00105         }
00106 
00107         private RasterImage FilterSRGBExact(RasterImage image, double standardDeviation)
00108         {
00109             IntPtr intermediateData = System.Runtime.InteropServices.Marshal.AllocHGlobal(image.Width
    * image.Height * (image.HasAlpha ?  4 :  3));
00110             IntPtr tbrData = System.Runtime.InteropServices.Marshal.AllocHGlobal(image.Width *
    image.Height * (image.HasAlpha ?  4 :  3));
00111             GC.AddMemoryPressure(2 * image.Width * image.Height * (image.HasAlpha ?  4 :  3));
00112 
00113             int width = image.Width;
00114             int height = image.Height;
00115 
00116             int kernelSize = (int)Math.Ceiling(standardDeviation * 3);
00117 
00118             double[] kernel = new double[kernelSize * 2 + 1];
00119 
00120             double total = 0;
00121 
00122             for (int i = 0; i <= kernelSize; i++)
00123             {
00124                 kernel[i] = Math.Exp(-(kernelSize - i) * (kernelSize - i) / (2 * standardDeviation *
    standardDeviation));
00125 
00126                 if (i < kernelSize)
00127                 {
00128                     kernel[2 * kernelSize - i] = kernel[i];
00129 
00130                     total += kernel[i] * 2;
00131                 }
00132                 else
00133                 {
00134                     total += kernel[i];
00135                 }
00136             }
00137 
00138             for (int i = 0; i < kernel.Length; i++)
00139             {
00140                 kernel[i] /= total;
00141             }
00142 
00143             int pixelSize = image.HasAlpha ?  4 :  3;
00144             int stride = image.Width * pixelSize;
00145 
00146             int threads;
00147 
00148             double size = Math.Sqrt((double)image.Width * image.Height);
00149 
00150             if (size <= 128)
00151             {
```

```
00152                 threads = 1;
00153             }
00154             else if (size <= 512)
00155             {
00156                 threads = Math.Min(4, Environment.ProcessorCount);
00157             }
00158             else
00159             {
00160                 threads = Math.Min(8, Environment.ProcessorCount);
00161             }
00162 
00163             unsafe
00164             {
00165                 byte* input = (byte*)image.ImageDataAddress;
00166                 byte* intermediate = (byte*)intermediateData;
00167                 byte* output = (byte*)tbrData;
00168 
00169                 Action<int> yLoop;
00170 
00171                 if (image.HasAlpha)
00172                 {
00173                     yLoop = y =>
00174                     {
00175                         for (int x = 0; x < width; x++)
00176                         {
00177                             double R = 0;
00178                             double G = 0;
00179                             double B = 0;
00180                             double weight = 0;
00181 
00182                             for (int targetX = 0; targetX <= kernelSize * 2; targetX++)
00183                             {
00184                                 int tX = Math.Min(Math.Max(0, x + targetX - kernelSize), width - 1);
00185 
00186                                 double a = input[y * stride + tX * 4 + 3] / 255.0;
00187 
00188                                 weight += kernel[targetX] * a;
00189 
00190                                 R += kernel[targetX] * input[y * stride + tX * 4] * a;
00191                                 G += kernel[targetX] * input[y * stride + tX * 4 + 1] * a;
00192                                 B += kernel[targetX] * input[y * stride + tX * 4 + 2] * a;
00193                             }
00194 
00195                             if (weight != 0)
00196                             {
00197                                 intermediate[y * stride + x * 4] = (byte)Math.Min(255, Math.Max(0, R /
      weight));
00198                                 intermediate[y * stride + x * 4 + 1] = (byte)Math.Min(255, Math.Max(0,
      G / weight));
00199                                 intermediate[y * stride + x * 4 + 2] = (byte)Math.Min(255, Math.Max(0,
      B / weight));
00200                                 intermediate[y * stride + x * 4 + 3] = (byte)Math.Min(255, Math.Max(0,
      weight * 255));
00201                             }
00202                             else
00203                             {
00204                                 intermediate[y * stride + x * 4] = 0;
00205                                 intermediate[y * stride + x * 4 + 1] = 0;
00206                                 intermediate[y * stride + x * 4 + 2] = 0;
00207                                 intermediate[y * stride + x * 4 + 3] = 0;
00208                             }
00209                         }
00210                     };
00211                 }
00212                 else
00213                 {
00214                     yLoop = y =>
00215                     {
00216                         for (int x = 0; x < width; x++)
00217                         {
00218                             double R = 0;
00219                             double G = 0;
00220                             double B = 0;
00221 
00222                             for (int targetX = 0; targetX <= kernelSize * 2; targetX++)
00223                             {
00224                                 int tX = Math.Min(Math.Max(0, x + targetX - kernelSize), width - 1);
00225 
00226                                 R += kernel[targetX] * input[y * stride + tX * 3];
00227                                 G += kernel[targetX] * input[y * stride + tX * 3 + 1];
00228                                 B += kernel[targetX] * input[y * stride + tX * 3 + 2];
00229                             }
00230 
00231                             intermediate[y * stride + x * 3] = (byte)Math.Min(255, Math.Max(0, R));
00232                             intermediate[y * stride + x * 3 + 1] = (byte)Math.Min(255, Math.Max(0,
      G));
00233                             intermediate[y * stride + x * 3 + 2] = (byte)Math.Min(255, Math.Max(0,
```

```
      B));
00234                         }
00235                     };
00236                 }
00237 
00238                 Action<int> xLoop;
00239 
00240                 if (image.HasAlpha)
00241                 {
00242                     xLoop = y =>
00243                     {
00244                         for (int x = 0; x < width; x++)
00245                         {
00246                             double R = 0;
00247                             double G = 0;
00248                             double B = 0;
00249 
00250 
00251                             double weight = 0;
00252 
00253                             for (int targetY = 0; targetY <= kernelSize * 2; targetY++)
00254                             {
00255                                 int tY = Math.Min(Math.Max(0, y + targetY - kernelSize), height - 1);
00256 
00257                                 double a = intermediate[tY * stride + x * 4 + 3] / 255.0;
00258 
00259                                 weight += kernel[targetY] * a;
00260 
00261                                 R += kernel[targetY] * intermediate[tY * stride + x * 4] * a;
00262                                 G += kernel[targetY] * intermediate[tY * stride + x * 4 + 1] * a;
00263                                 B += kernel[targetY] * intermediate[tY * stride + x * 4 + 2] * a;
00264                             }
00265 
00266                             if (weight != 0)
00267                             {
00268                                 output[y * stride + x * 4] = (byte)Math.Min(255, Math.Max(0, R /
      weight));
00269                                 output[y * stride + x * 4 + 1] = (byte)Math.Min(255, Math.Max(0, G /
      weight));
00270                                 output[y * stride + x * 4 + 2] = (byte)Math.Min(255, Math.Max(0, B /
      weight));
00271                                 output[y * stride + x * 4 + 3] = (byte)Math.Min(255, Math.Max(0,
      weight * 255));
00272                             }
00273                             else
00274                             {
00275                                 output[y * stride + x * 4] = 0;
00276                                 output[y * stride + x * 4 + 1] = 0;
00277                                 output[y * stride + x * 4 + 2] = 0;
00278                                 output[y * stride + x * 4 + 3] = 0;
00279                             }
00280                         }
00281                     };
00282                 }
00283                 else
00284                 {
00285                     xLoop = y =>
00286                     {
00287                         for (int x = 0; x < width; x++)
00288                         {
00289                             double R = 0;
00290                             double G = 0;
00291                             double B = 0;
00292 
00293                             for (int targetY = 0; targetY <= kernelSize * 2; targetY++)
00294                             {
00295                                 int tY = Math.Min(Math.Max(0, y + targetY - kernelSize), height - 1);
00296 
00297                                 R += kernel[targetY] * intermediate[tY * stride + x * 3];
00298                                 G += kernel[targetY] * intermediate[tY * stride + x * 3 + 1];
00299                                 B += kernel[targetY] * intermediate[tY * stride + x * 3 + 2];
00300                             }
00301 
00302                             output[y * stride + x * 3] = (byte)Math.Min(255, Math.Max(0, R));
00303                             output[y * stride + x * 3 + 1] = (byte)Math.Min(255, Math.Max(0, G));
00304                             output[y * stride + x * 3 + 2] = (byte)Math.Min(255, Math.Max(0, B));
00305                         }
00306                     };
00307                 }
00308 
00309 
00310 
00311                 if (threads == 1)
00312                 {
00313                     for (int y = 0; y < height; y++)
00314                     {
00315                         yLoop(y);
```

```
00316                     }
00317
00318                     for (int y = 0; y < height; y++)
00319                     {
00320                         xLoop(y);
00321                     }
00322                 }
00323                 else
00324                 {
00325                     ParallelOptions options = new ParallelOptions() { MaxDegreeOfParallelism = threads
      };
00326
00327                     Parallel.For(0, height, options, yLoop);
00328                     Parallel.For(0, height, options, xLoop);
00329                 }
00330             }
00331
00332             System.Runtime.InteropServices.Marshal.FreeHGlobal(intermediateData);
00333             GC.RemoveMemoryPressure(image.Width * image.Height * (image.HasAlpha ?  4 :  3));
00334
00335             DisposableIntPtr disp = new DisposableIntPtr(tbrData);
00336
00337             return new RasterImage(ref disp, image.Width, image.Height, image.HasAlpha,
      image.Interpolate);
00338         }
00339
00340     }
00341 }
00342
```

8.78 MaskFilter.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Diagnostics.Contracts;
00020 using System.Threading.Tasks;
00021
00022 namespace VectSharp.Filters
00023 {
00024 /// <summary>
00025 /// Represents a filter that uses the luminance of an image to mask another image.
00026 /// </summary>
00027     public class MaskFilter :  FilterWithRasterisableParameter, IFilterWithLocation
00028     {
00029 /// <inheritdoc/>
00030         public Point TopLeftMargin => new Point(0, 0);
00031 /// <inheritdoc/>
00032         public Point BottomRightMargin => new Point(0, 0);
00033
00034 /// <summary>
00035 /// The image that is used to mask the input image.
00036 /// </summary>
00037         public Graphics Mask => this.RasterisableParameter;
00038
00039 /// <summary>
00040 /// Creates a new <see cref="MaskFilter"/> with the specified mask image.
00041 /// </summary>
00042 /// <param name="mask">The image that is used to mask the input image.</param>
00043         public MaskFilter(Graphics mask) :  base(mask) { }
00044
00045 /// <inheritdoc/>
00046         [Pure]
00047         public RasterImage Filter(RasterImage image, Rectangle bounds, double scale)
00048         {
00049             RasterImage mask = this.GetCachedRasterisation(scale);
00050
00051             IntPtr tbrData = System.Runtime.InteropServices.Marshal.AllocHGlobal(image.Width *
      image.Height * 4);
```

```
00052             GC.AddMemoryPressure(image.Width * image.Height * 4);
00053
00054             int width = image.Width;
00055             int height = image.Height;
00056
00057             int pixelSizeInput = image.HasAlpha ?  4 :  3;
00058             int strideInput = image.Width * pixelSizeInput;
00059
00060             int pixelSizeOutput = 4;
00061             int strideOutput = image.Width * pixelSizeOutput;
00062
00063             int pixelSizeMask = 4;
00064             int strideMask = mask.Width * pixelSizeMask;
00065
00066             int maskWidth = mask.Width;
00067             int maskHeight = mask.Height;
00068
00069             int threads = Math.Min(8, Environment.ProcessorCount);
00070
00071             unsafe
00072             {
00073                 byte* input = (byte*)image.ImageDataAddress;
00074                 byte* maskBytes = (byte*)mask.ImageDataAddress;
00075                 byte* output = (byte*)tbrData;
00076
00077                 Action<int> yLoop;
00078
00079                 if (image.HasAlpha)
00080                 {
00081                     yLoop = (y) =>
00082                     {
00083                         for (int x = 0; x < width; x++)
00084                         {
00085                             int maskX = (int)Math.Round((bounds.Location.X + (x + 0.5) *
      bounds.Size.Width / width - CachedBounds.Location.X) / CachedBounds.Size.Width * maskWidth);
00086                             int maskY = (int)Math.Round((bounds.Location.Y + (y + 0.5) *
      bounds.Size.Height / height - CachedBounds.Location.Y) / CachedBounds.Size.Height * maskHeight);
00087
00088                             double weight = 0;
00089
00090                             if (maskX >= 0 && maskX < maskWidth && maskY >= 0 && maskY < maskHeight)
00091                             {
00092                                 weight = (maskBytes[maskY * strideMask + maskX * pixelSizeMask] *
      0.2126 + maskBytes[maskY * strideMask + maskX * pixelSizeMask + 1] * 0.7152 + maskBytes[maskY *
      strideMask + maskX * pixelSizeMask + 2] * 0.0722) / 255.0;
00093
00094                                 if (mask.HasAlpha)
00095                                 {
00096                                     weight *= maskBytes[maskY * strideMask + maskX * pixelSizeMask +
      3] / 255.0;
00097                                 }
00098                             }
00099
00100                             if (weight > 0)
00101                             {
00102                                 output[y * strideOutput + x * 4] = input[y * strideInput + x * 4];
00103                                 output[y * strideOutput + x * 4 + 1] = input[y * strideInput + x * 4 +
      1];
00104                                 output[y * strideOutput + x * 4 + 2] = input[y * strideInput + x * 4 +
      2];
00105                                 output[y * strideOutput + x * 4 + 3] = (byte)Math.Round(input[y *
      strideInput + x * 4 + 3] * weight);
00106                             }
00107                             else
00108                             {
00109                                 output[y * strideOutput + x * 4] = 0;
00110                                 output[y * strideOutput + x * 4 + 1] = 0;
00111                                 output[y * strideOutput + x * 4 + 2] = 0;
00112                                 output[y * strideOutput + x * 4 + 3] = 0;
00113                             }
00114                         }
00115                     };
00116                 }
00117                 else
00118                 {
00119                     yLoop = (y) =>
00120                     {
00121                         for (int x = 0; x < width; x++)
00122                         {
00123                             int maskX = (int)Math.Round((bounds.Location.X + (x + 0.5) *
      bounds.Size.Width / width - CachedBounds.Location.X) / CachedBounds.Size.Width * maskWidth);
00124                             int maskY = (int)Math.Round((bounds.Location.Y + (y + 0.5) *
      bounds.Size.Height / height - CachedBounds.Location.Y) / CachedBounds.Size.Height * maskHeight);
00125
00126                             double weight = 0;
00127
00128                             if (maskX >= 0 && maskX < maskWidth && maskY >= 0 && maskY < maskHeight)
```

```
00129                             {
00130                                 weight = (maskBytes[maskY * strideMask + maskX * pixelSizeMask] *
      0.2126 + maskBytes[maskY * strideMask + maskX * pixelSizeMask + 1] * 0.7152 + maskBytes[maskY *
      strideMask + maskX * pixelSizeMask + 2] * 0.0722);
00131
00132                                 if (mask.HasAlpha)
00133                                 {
00134                                     weight *= maskBytes[maskY * strideMask + maskX * pixelSizeMask +
      3] / 255.0;
00135                                 }
00136                             }
00137
00138                             if (weight > 0)
00139                             {
00140                                 output[y * strideOutput + x * 4] = input[y * strideInput + x * 3];
00141                                 output[y * strideOutput + x * 4 + 1] = input[y * strideInput + x * 3 +
      1];
00142                                 output[y * strideOutput + x * 4 + 2] = input[y * strideInput + x * 3 +
      2];
00143                                 output[y * strideOutput + x * 4 + 3] = (byte)Math.Round(weight);
00144                             }
00145                             else
00146                             {
00147                                 output[y * strideOutput + x * 4] = 0;
00148                                 output[y * strideOutput + x * 4 + 1] = 0;
00149                                 output[y * strideOutput + x * 4 + 2] = 0;
00150                                 output[y * strideOutput + x * 4 + 3] = 0;
00151                             }
00152                         }
00153                     };
00154                 }
00155
00156                 if (threads == 1)
00157                 {
00158                     for (int y = 0; y < height; y++)
00159                     {
00160                         yLoop(y);
00161                     }
00162                 }
00163                 else
00164                 {
00165                     ParallelOptions options = new ParallelOptions() { MaxDegreeOfParallelism = threads
      };
00166
00167                     Parallel.For(0, height, options, yLoop);
00168                 }
00169
00170             }
00171
00172             DisposableIntPtr disp = new DisposableIntPtr(tbrData);
00173
00174             return new RasterImage(ref disp, width, height, image.HasAlpha, image.Interpolate);
00175
00176         }
00177     }
00178 }
```

8.79 Font.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Diagnostics;
00021 using System.IO;
00022
00023 namespace VectSharp
00024 {
```

```
00025 /// <summary>
00026 /// Represents a typeface with a specific size.
00027 /// </summary>
00028     [DebuggerDisplay("\\{ FontSize:  {FontSize}, FontFamily:  {FontFamily} \\}")]
00029     public class Font
00030     {
00031 /// <summary>
00032 /// Determines whether text kerning is enabled.  Note that, even when this is set to <see
      langword="false" />, text kerning will be applied on some platforms.  For the best consistency, leave
      this set to <see langword="true"/>.
00033 /// </summary>
00034         public static bool EnableKerning = true;
00035
00036 /// <summary>
00037 /// Represents options to underline text.
00038 /// </summary>
00039         public class FontUnderline
00040         {
00041 /// <summary>
00042 /// Determines whether the underline skips the parts of the glyph that would intersect with it or not.
00043 /// </summary>
00044             public bool SkipDescenders { get; set; }
00045
00046 /// <summary>
00047 /// Determines the position of the top of the underline with respect to the text baseline.  Positive
      values are below the baseline, negative values are above it.  This is expressed as a fraction of the
      font size.
00048 /// </summary>
00049             public double Position { get; set; }
00050
00051 /// <summary>
00052 /// Determines the thickness of the underline, expressed as a fraction of the font size.
00053 /// </summary>
00054             public double Thickness { get; set; }
00055
00056 /// <summary>
00057 /// Determines the caps at the start and end of the underline.
00058 /// </summary>
00059             public LineCaps LineCap { get; set; }
00060
00061 /// <summary>
00062 /// Determine whether the shape of the underline is slanted to follow the angle of italic fonts.
00063 /// </summary>
00064             public bool FollowItalicAngle { get; set; }
00065
00066 /// <summary>
00067 /// Create a new underline with the default settings for the specified font family.
00068 /// </summary>
00069 /// <param name="family">The font family to which the underline will be applied.</param>
00070             internal FontUnderline(FontFamily family)
00071             {
00072                 this.SkipDescenders = true;
00073                 this.LineCap = LineCaps.Butt;
00074                 this.FollowItalicAngle = true;
00075
00076                 if (family.TrueTypeFile != null)
00077                 {
00078                     this.Position = -family.TrueTypeFile.Get1000EmUnderlinePosition() / 1000.0;
00079                     this.Thickness = family.TrueTypeFile.Get1000EmUnderlineThickness() / 1000.0;
00080                 }
00081                 else
00082                 {
00083                     this.Position = 0.3;
00084                     this.Thickness = family.IsBold ?  0.15 :  0.075;
00085                 }
00086
00087                 if (double.IsNaN(this.Position))
00088                 {
00089                     this.Position = 0.3;
00090                 }
00091
00092                 if (double.IsNaN(this.Thickness))
00093                 {
00094                     this.Thickness = family.IsBold ?  0.15 :  0.075;
00095                 }
00096             }
00097         }
00098
00099 /// <summary>
00100 /// Represents detailed information about the metrics of a text string when drawn with a certain font.
00101 /// </summary>
00102         public class DetailedFontMetrics
00103         {
00104 /// <summary>
00105 /// Width of the text (measured on the actual glyph outlines).
00106 /// </summary>
00107             public double Width { get; }
```

```
00108
00109 /// <summary>
00110 /// Height of the text (measured on the actual glyph outlines).
00111 /// </summary>
00112             public double Height { get; }
00113
00114 /// <summary>
00115 /// How much the leftmost glyph in the string overhangs the glyph origin on the left.  Positive for
      glyphs that hang past the origin (e.g.  italic 'f').
00116 /// </summary>
00117             public double LeftSideBearing { get; }
00118
00119 /// <summary>
00120 /// How much the rightmost glyph in the string overhangs the glyph end on the right.  Positive for
      glyphs that hang past the end (e.g.  italic 'f').
00121 /// </summary>
00122             public double RightSideBearing { get; }
00123
00124 /// <summary>
00125 /// Height of the tallest glyph in the string over the baseline.  Always &gt;= 0.
00126 /// </summary>
00127             public double Top { get; }
00128
00129 /// <summary>
00130 /// Depth of the deepest glyph in the string below the baseline.  Always &lt;= 0.
00131 /// </summary>
00132             public double Bottom { get; }
00133
00134 /// <summary>
00135 /// Advance width of the text (excluding any left- or right- side bearing).
00136 /// </summary>
00137             public double AdvanceWidth { get; }
00138
00139             internal DetailedFontMetrics(double width, double height, double leftSideBearing, double
      rightSideBearing, double top, double bottom, double advanceWidth)
00140             {
00141                 this.Width = width;
00142                 this.Height = height;
00143                 this.LeftSideBearing = leftSideBearing;
00144                 this.RightSideBearing = rightSideBearing;
00145                 this.Top = top;
00146                 this.Bottom = bottom;
00147                 this.AdvanceWidth = advanceWidth;
00148             }
00149         }
00150
00151 /// <summary>
00152 /// Font size, in graphics units.
00153 /// </summary>
00154         public double FontSize { get; }
00155
00156 /// <summary>
00157 /// Font typeface.
00158 /// </summary>
00159         public FontFamily FontFamily { get; }
00160
00161 /// <summary>
00162 /// Create a new Font object, given the base typeface and the font size.
00163 /// </summary>
00164 /// <param name="fontFamily">Base typeface.  See <see cref="FontFamily"/>.</param>
00165 /// <param name="fontSize">The font size, in graphics units.</param>
00166         public Font(FontFamily fontFamily, double fontSize)
00167         {
00168             this.FontFamily = fontFamily;
00169             this.FontSize = fontSize;
00170         }
00171
00172 /// <summary>
00173 /// Create a new Font object, given the base typeface, the font size, and a boolean value determining
      whether text using this font should be underlined.
00174 /// </summary>
00175 /// <param name="fontFamily">Base typeface.  See <see cref="FontFamily"/>.</param>
00176 /// <param name="fontSize">The font size, in graphics units.</param>
00177 /// <param name="underlined">A boolean value determining whether text drawn using this font should be
      underlined.  The appearance of the underline can be tweaked by changing the properties of the <see
      cref="Underline"/> property after the font has been created.</param>
00178         public Font(FontFamily fontFamily, double fontSize, bool underlined)
00179         {
00180             this.FontFamily = fontFamily;
00181             this.FontSize = fontSize;
00182
00183             if (underlined)
00184             {
00185                 this.Underline = new FontUnderline(fontFamily);
00186             }
00187         }
00188
```

```
00189 /// <summary>
00190 /// Create a new Font object, given the base typeface, the font size, and an object describing the
      underline properties of text drawn using this font.
00191 /// </summary>
00192 /// <param name="fontFamily">Base typeface.  See <see cref="FontFamily"/>.</param>
00193 /// <param name="fontSize">The font size, in graphics units.</param>
00194 /// <param name="underline">A <see cref="FontUnderline"/> object describing the underline properties
      of text drawn using this font.</param>
00195         public Font(FontFamily fontFamily, double fontSize, FontUnderline underline)
00196         {
00197             this.FontFamily = fontFamily;
00198             this.FontSize = fontSize;
00199             this.Underline = underline;
00200         }
00201 
00202 /// <summary>
00203 /// Maximum height over the baseline of the usual glyphs in the font (there may be glyphs taller than
      this).  Always &gt;= 0.
00204 /// </summary>
00205         public double Ascent
00206         {
00207             get
00208             {
00209                 if (this.FontFamily.TrueTypeFile == null)
00210                 {
00211                     return 0;
00212                 }
00213                 else
00214                 {
00215                     return this.FontFamily.TrueTypeFile.Get1000EmAscent() * this.FontSize / 1000;
00216                 }
00217             }
00218         }
00219 
00220 /// <summary>
00221 /// Height above the baseline for a clipping region (Windows ascent).  Always &gt;= 0.
00222 /// </summary>
00223         public double WinAscent
00224         {
00225             get
00226             {
00227                 if (this.FontFamily.TrueTypeFile == null)
00228                 {
00229                     return 0;
00230                 }
00231                 else
00232                 {
00233                     return this.FontFamily.TrueTypeFile.Get1000EmWinAscent() * this.FontSize / 1000;
00234                 }
00235             }
00236         }
00237 
00238 /// <summary>
00239 /// Maximum depth below the baseline of the usual glyphs in the font (there may be glyphs deeper than
      this).  Always &lt;= 0.
00240 /// </summary>
00241         public double Descent
00242         {
00243             get
00244             {
00245                 if (this.FontFamily.TrueTypeFile == null)
00246                 {
00247                     return 0;
00248                 }
00249                 else
00250                 {
00251                     return this.FontFamily.TrueTypeFile.Get1000EmDescent() * this.FontSize / 1000;
00252                 }
00253             }
00254         }
00255 
00256 /// <summary>
00257 /// Absolute maximum height over the baseline of the glyphs in the font.  Always &gt;= 0.
00258 /// </summary>
00259         public double YMax
00260         {
00261             get
00262             {
00263                 if (this.FontFamily.TrueTypeFile == null)
00264                 {
00265                     return 0;
00266                 }
00267                 else
00268                 {
00269                     return this.FontFamily.TrueTypeFile.Get1000EmYMax() * this.FontSize / 1000;
00270                 }
00271             }
```

```
00272         }
00273
00274 /// <summary>
00275 /// Absolute maximum depth below the baseline of the glyphs in the font.  Always &lt;= 0.
00276 /// </summary>
00277         public double YMin
00278         {
00279             get
00280             {
00281                 if (this.FontFamily.TrueTypeFile == null)
00282                 {
00283                     return 0;
00284                 }
00285                 else
00286                 {
00287                     return this.FontFamily.TrueTypeFile.Get1000EmYMin() * this.FontSize / 1000;
00288                 }
00289             }
00290         }
00291
00292 /// <summary>
00293 /// Determines the underline style of text drawn using this font.  If this is <see langword="null"/>,
      the text is not underlined.
00294 /// </summary>
00295         public FontUnderline Underline { get; }
00296
00297 /// <summary>
00298 /// Measure the size of a text string when typeset with this font.
00299 /// </summary>
00300 /// <param name="text">The string to measure.</param>
00301 /// <returns>A <see cref="Size"/> object representing the width and height of the text.</returns>
00302         public Size MeasureText(string text)
00303         {
00304             if (this.FontFamily.TrueTypeFile != null && !string.IsNullOrEmpty(text))
00305             {
00306                 double width = 0;
00307                 double yMin = 0;
00308                 double yMax = 0;
00309
00310                 Point currentGlyphPlacementDelta = new Point();
00311                 Point currentGlyphAdvanceDelta = new Point();
00312                 Point nextGlyphPlacementDelta = new Point();
00313                 Point nextGlyphAdvanceDelta = new Point();
00314
00315                 for (int i = 0; i < text.Length; i++)
00316                 {
00317                     if (Font.EnableKerning && i < text.Length - 1)
00318                     {
00319                         currentGlyphPlacementDelta = nextGlyphPlacementDelta;
00320                         currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
00321                         nextGlyphAdvanceDelta = new Point();
00322                         nextGlyphPlacementDelta = new Point();
00323
00324                         TrueTypeFile.PairKerning kerning =
      this.FontFamily.TrueTypeFile.Get1000EmKerning(text[i], text[i + 1]);
00325
00326                         if (kerning != null)
00327                         {
00328                             currentGlyphPlacementDelta = new Point(currentGlyphPlacementDelta.X +
      kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y + kerning.Glyph1Placement.Y);
00329                             currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
      kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
00330
00331                             nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
      kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
00332                             nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
      kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
00333                         }
00334                     }
00335
00336                     width += (this.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(text[i]) +
      currentGlyphAdvanceDelta.X) * this.FontSize / 1000;
00337                     TrueTypeFile.VerticalMetrics vMet =
      this.FontFamily.TrueTypeFile.Get1000EmGlyphVerticalMetrics(text[i]);
00338
00339                     yMin = Math.Min(yMin, (vMet.YMin + currentGlyphPlacementDelta.Y) * this.FontSize /
      1000);
00340                     yMax = Math.Max(yMax, (vMet.YMax + currentGlyphPlacementDelta.Y) * this.FontSize /
      1000);
00341                 }
00342
00343                 width -= this.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(text[0]).LeftSideBearing
      * this.FontSize / 1000;
00344                 width -= this.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(text[text.Length -
      1]).RightSideBearing * this.FontSize / 1000;
00345
00346                 return new Size(width, yMax - yMin);
```

```
00347             }
00348             else
00349             {
00350                 return new Size(0, 0);
00351             }
00352         }
00353 
00354 /// <summary>
00355 /// Measure all the metrics of a text string when typeset with this font.
00356 /// </summary>
00357 /// <param name="text">The string to measure.</param>
00358 /// <returns>A <see cref="DetailedFontMetrics"/> object representing the metrics of the
    text.</returns>
00359         public DetailedFontMetrics MeasureTextAdvanced(string text)
00360         {
00361             if (this.FontFamily.TrueTypeFile != null && !string.IsNullOrEmpty(text))
00362             {
00363                 double width = 0;
00364                 double yMin = 0;
00365                 double yMax = 0;
00366 
00367                 Point currentGlyphPlacementDelta = new Point();
00368                 Point currentGlyphAdvanceDelta = new Point();
00369                 Point nextGlyphPlacementDelta = new Point();
00370                 Point nextGlyphAdvanceDelta = new Point();
00371 
00372                 for (int i = 0; i < text.Length; i++)
00373                 {
00374                     currentGlyphPlacementDelta = nextGlyphPlacementDelta;
00375                     currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
00376 
00377                     if (Font.EnableKerning && i < text.Length - 1)
00378                     {
00379                         nextGlyphAdvanceDelta = new Point();
00380                         nextGlyphPlacementDelta = new Point();
00381 
00382                         TrueTypeFile.PairKerning kerning =
    this.FontFamily.TrueTypeFile.Get1000EmKerning(text[i], text[i + 1]);
00383 
00384                         if (kerning != null)
00385                         {
00386                             currentGlyphPlacementDelta = new Point(currentGlyphPlacementDelta.X +
    kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y + kerning.Glyph1Placement.Y);
00387                             currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
    kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
00388 
00389                             nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
    kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
00390                             nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
    kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
00391                         }
00392                     }
00393 
00394                     width += (this.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(text[i]) +
    currentGlyphAdvanceDelta.X) * this.FontSize / 1000;
00395                     TrueTypeFile.VerticalMetrics vMet =
    this.FontFamily.TrueTypeFile.Get1000EmGlyphVerticalMetrics(text[i]);
00396 
00397                     yMin = Math.Min(yMin, (vMet.YMin + currentGlyphPlacementDelta.Y) * this.FontSize /
    1000);
00398                     yMax = Math.Max(yMax, (vMet.YMax + currentGlyphPlacementDelta.Y) * this.FontSize /
    1000);
00399                 }
00400 
00401                 double lsb =
    this.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(text[0]).LeftSideBearing * this.FontSize / 1000;
00402                 double rsb = this.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(text[text.Length -
    1]).RightSideBearing * this.FontSize / 1000;
00403 
00404                 double advanceWidth = width;
00405 
00406                 width -= lsb;
00407                 width -= rsb;
00408 
00409                 return new DetailedFontMetrics(width, yMax - yMin, lsb, rsb, yMax, yMin,
    advanceWidth);
00410             }
00411             else
00412             {
00413                 return new DetailedFontMetrics(0, 0, 0, 0, 0, 0, 0);
00414             }
00415         }
00416     }
00417 
00418 
00419 /// <summary>
00420 /// Represents a typeface.
```

```
00421 /// </summary>
00422     [DebuggerDisplay("\\{ FamilyName:  {FamilyName}, IsBold:  {IsBold}, IsItalic:  {IsItalic},
      IsOblique:  {IsOblique} \\}")]
00423     public class FontFamily
00424     {
00425 /// <summary>
00426 /// The default font library used to resolve font family names.
00427 /// </summary>
00428         public static IFontLibrary DefaultFontLibrary { get; set; } = new DefaultFontLibrary();
00429
00430 /// <summary>
00431 /// Create a new font family from the specified family name or true type file.  If the family name or
      the true type file are not valid, an exception might be raised.  Equivalent to
      DefaultFontLibrary.ResolveFontFamily.
00432 /// </summary>
00433 /// <param name="fontFamily">The name of the font family to create, or the path to a TTF file.</param>
00434 /// <returns>If the font family name or the true type file is valid, a <see cref="FontFamily"/> object
      corresponding to the specified font family.</returns>
00435         public static FontFamily ResolveFontFamily(string fontFamily) =>
      DefaultFontLibrary.ResolveFontFamily(fontFamily);
00436
00437
00438 /// <summary>
00439 /// Create a new font family from the specified standard font family name.  Equivalent to
      DefaultFontLibrary.ResolveFontFamily.
00440 /// </summary>
00441 /// <param name="standardFontFamily">The standard name of the font family.</param>
00442 /// <returns>A <see cref="FontFamily"/> object corresponding to the specified font family.</returns>
00443         public static FontFamily ResolveFontFamily(StandardFontFamilies standardFontFamily) =>
      DefaultFontLibrary.ResolveFontFamily(standardFontFamily);
00444
00445 /// <summary>
00446 /// Create a new font family from the specified family name or true type file.  If the family name or
      the true type file are not valid, try to instantiate the font family using
00447 /// the <paramref name="fallback"/>.  If none of the fallback family names or true type files are
      valid, an exception might be raised.  Equivalent to DefaultFontLibrary.ResolveFontFamily.
00448 /// </summary>
00449 /// <param name="fontFamily">The name of the font family to create, or the path to a TTF file.</param>
00450 /// <param name="fallback">Names of additional font families or TTF files, which will be tried if the
      first <paramref name="fontFamily"/> is not valid.</param>
00451 /// <returns>A <see cref="FontFamily"/> object corresponding to the first of the specified font
      families that is valid.</returns>
00452         public static FontFamily ResolveFontFamily(string fontFamily, params string[] fallback) =>
      DefaultFontLibrary.ResolveFontFamily(fontFamily, fallback);
00453
00454 /// <summary>
00455 /// Create a new font family from the specified family name or true type file.  If the family name or
      the true type file are not valid, try to instantiate the font family using
00456 /// the <paramref name="fallback"/>.  If none of the fallback family names or true type files are
      valid, instantiate a standard font family using the <paramref name="finalFallback"/>.  Equivalent to
      DefaultFontLibrary.ResolveFontFamily.
00457 /// </summary>
00458 /// <param name="fontFamily">The name of the font family to create, or the path to a TTF file.</param>
00459 /// <param name="fallback">Names of additional font families or TTF files, which will be tried if the
      first <paramref name="fontFamily"/> is not valid.</param>
00460 /// <param name="finalFallback">The standard name of the font family that will be used if none of the
      fallback families are valid.</param>
00461 /// <returns>A <see cref="FontFamily"/> object corresponding to the first of the specified font
      families that is valid.</returns>
00462         public static FontFamily ResolveFontFamily(string fontFamily, StandardFontFamilies
      finalFallback, params string[] fallback) => DefaultFontLibrary.ResolveFontFamily(fontFamily,
      finalFallback, fallback);
00463
00464         internal static object fontFamilyLock = new object();
00465         internal static readonly Dictionary<string, Stream> manifestResources = new Dictionary<string,
      Stream>();
00466
00467         internal static Stream GetManifestResourceStream(string name)
00468         {
00469             if (!manifestResources.ContainsKey(name))
00470             {
00471                 manifestResources.Add(name,
      typeof(FontFamily).Assembly.GetManifestResourceStream(name));
00472             }
00473
00474             return manifestResources[name];
00475         }
00476
00477
00478 /// <summary>
00479 /// The names of the 14 standard families that are guaranteed to be displayed correctly.
00480 /// </summary>
00481         public static string[] StandardFamilies = new string[] { "Times-Roman", "Times-Bold",
      "Times-Italic", "Times-BoldItalic", "Helvetica", "Helvetica-Bold", "Helvetica-Oblique",
      "Helvetica-BoldOblique", "Courier", "Courier-Bold", "Courier-Oblique", "Courier-BoldOblique",
      "Symbol", "ZapfDingbats" };
00482
```

```
00483 /// <summary>
00484 /// The names of the resource streams pointing to the included TrueType font files for each of the
      standard 14 font families.
00485 /// </summary>
00486         public static string[] StandardFontFamilyResources = new string[]
00487         {
00488             "VectSharp.StandardFonts.Tinos-Regular.ttf", "VectSharp.StandardFonts.Tinos-Bold.ttf",
      "VectSharp.StandardFonts.Tinos-Italic.ttf", "VectSharp.StandardFonts.Tinos-BoldItalic.ttf",
00489             "VectSharp.StandardFonts.Arimo-Regular.ttf", "VectSharp.StandardFonts.Arimo-Bold.ttf",
      "VectSharp.StandardFonts.Arimo-Italic.ttf", "VectSharp.StandardFonts.Arimo-BoldItalic.ttf",
00490             "VectSharp.StandardFonts.Cousine-Regular.ttf", "VectSharp.StandardFonts.Cousine-Bold.ttf",
      "VectSharp.StandardFonts.Cousine-Italic.ttf", "VectSharp.StandardFonts.Cousine-BoldItalic.ttf",
00491             "VectSharp.StandardFonts.SymbolNeu_GB.ttf",
      "VectSharp.StandardFonts.Levibats-Regular_GB.ttf"
00492         };
00493
00494 /// <summary>
00495 /// Whether this is one of the 14 standard font families or not.
00496 /// </summary>
00497         public bool IsStandardFamily { get; internal set; }
00498
00499 /// <summary>
00500 /// The 14 standard font families.
00501 /// </summary>
00502         public enum StandardFontFamilies
00503         {
00504 /// <summary>
00505 /// Serif normal regular face.
00506 /// </summary>
00507             TimesRoman,
00508
00509 /// <summary>
00510 /// Serif bold regular face.
00511 /// </summary>
00512             TimesBold,
00513
00514 /// <summary>
00515 /// Serif normal italic face.
00516 /// </summary>
00517             TimesItalic,
00518
00519 /// <summary>
00520 /// Serif bold italic face.
00521 /// </summary>
00522             TimesBoldItalic,
00523
00524 /// <summary>
00525 /// Sans-serif normal regular face.
00526 /// </summary>
00527             Helvetica,
00528
00529 /// <summary>
00530 /// Sans-serif bold regular face.
00531 /// </summary>
00532             HelveticaBold,
00533
00534 /// <summary>
00535 /// Sans-serif normal oblique face.
00536 /// </summary>
00537             HelveticaOblique,
00538
00539 /// <summary>
00540 /// Sans-serif bold oblique face.
00541 /// </summary>
00542             HelveticaBoldOblique,
00543
00544 /// <summary>
00545 /// Monospace normal regular face.
00546 /// </summary>
00547             Courier,
00548
00549 /// <summary>
00550 /// Monospace bold regular face.
00551 /// </summary>
00552             CourierBold,
00553
00554 /// <summary>
00555 /// Monospace normal oblique face.
00556 /// </summary>
00557             CourierOblique,
00558
00559 /// <summary>
00560 /// Monospace bold oblique face.
00561 /// </summary>
00562             CourierBoldOblique,
00563
00564 /// <summary>
```

```
00565 /// Symbol font.
00566 /// </summary>
00567             Symbol,
00568 
00569 /// <summary>
00570 /// Dingbat font.
00571 /// </summary>
00572             ZapfDingbats
00573         }
00574 
00575 /// <summary>
00576 /// Full path to the TrueType font file for this font family (or, if this is a standard font family,
      name of the font family).
00577 /// </summary>
00578         public string FileName { get; internal set; }
00579 
00580 /// <summary>
00581 /// Name of the font family, including any variantes.
00582 /// </summary>
00583         public string FamilyName { get; internal set; }
00584 
00585 /// <summary>
00586 /// Parsed TrueType font file for this font family.
00587 /// See also:  <seealso cref="VectSharp.TrueTypeFile"/>.
00588 /// </summary>
00589         public TrueTypeFile TrueTypeFile { get; }
00590 
00591 /// <summary>
00592 /// Whether this font is bold or not.  This is set based on the information included in the OS/2 table
      of the TrueType file.
00593 /// </summary>
00594         public bool IsBold { get; internal set; }
00595 
00596 /// <summary>
00597 /// Whether this font is italic or oblique or not.  This is set based on the information included in
      the OS/2 table of the TrueType file.
00598 /// </summary>
00599         public bool IsItalic { get; internal set; }
00600 
00601 /// <summary>
00602 /// Whether this font is oblique or not.  This is set based on the information included in the OS/2
      table of the TrueType file.
00603 /// </summary>
00604         public bool IsOblique { get; internal set; }
00605 
00606 /// <summary>
00607 /// Create a new <see cref="FontFamily"/>.
00608 /// </summary>
00609 /// <param name="fileName">The full path to the TrueType font file for this font family or the name of
      a standard font family.</param>
00610         [Obsolete("Please use the FontFamily.ResolveFontFamily(string) method instead!", true)]
00611         public FontFamily(string fileName)
00612         {
00613             lock (fontFamilyLock)
00614             {
00615                 FontFamily resolved = DefaultFontLibrary.ResolveFontFamily(fileName);
00616 
00617                 this.FileName = resolved.FileName;
00618                 this.FamilyName = resolved.FamilyName;
00619                 this.TrueTypeFile = resolved.TrueTypeFile;
00620                 this.IsOblique = resolved.IsOblique;
00621                 this.IsBold = resolved.IsBold;
00622                 this.IsItalic = resolved.IsItalic;
00623                 this.IsStandardFamily = resolved.IsStandardFamily;
00624             }
00625         }
00626 
00627         internal FontFamily()
00628         {
00629 
00630         }
00631 
00632 /// <summary>
00633 /// Create a new <see cref="FontFamily"/>.
00634 /// </summary>
00635 /// <param name="ttfStream">A stream containing a file in TTF format.</param>
00636         public FontFamily(Stream ttfStream)
00637         {
00638             lock (fontFamilyLock)
00639             {
00640                 IsStandardFamily = false;
00641 
00642                 TrueTypeFile = TrueTypeFile.CreateTrueTypeFile(ttfStream);
00643 
00644                 FileName = TrueTypeFile.GetFontFamilyName();
00645                 FamilyName = TrueTypeFile.GetFullFontFamilyName() ??  FileName;
00646                 this.IsBold = TrueTypeFile.IsBold();
```

```
00647                 this.IsItalic = TrueTypeFile.IsItalic();
00648                 this.IsOblique = TrueTypeFile.IsOblique();
00649             }
00650         }
00651 
00652 /// <summary>
00653 /// Create a new <see cref="FontFamily"/>.
00654 /// </summary>
00655 /// <param name="ttf">A font file in TTF format.</param>
00656         public FontFamily(TrueTypeFile ttf)
00657         {
00658             lock (fontFamilyLock)
00659             {
00660                 IsStandardFamily = false;
00661 
00662                 TrueTypeFile = ttf;
00663 
00664                 FileName = TrueTypeFile.GetFontFamilyName();
00665                 FamilyName = TrueTypeFile.GetFullFontFamilyName() ??  FileName;
00666                 this.IsBold = TrueTypeFile.IsBold();
00667                 this.IsItalic = TrueTypeFile.IsItalic();
00668                 this.IsOblique = TrueTypeFile.IsOblique();
00669             }
00670         }
00671 
00672 /// <summary>
00673 /// Create a new standard <see cref="FontFamily"/>.
00674 /// </summary>
00675 /// <param name="standardFontFamily">The standard font family.</param>
00676         [Obsolete("Please use the FontFamily.ResolveFontFamily(StandardFontFamilies) method instead!",
      true)]
00677         public FontFamily(StandardFontFamilies standardFontFamily)
00678         {
00679             lock (fontFamilyLock)
00680             {
00681                 FontFamily resolved = DefaultFontLibrary.ResolveFontFamily(standardFontFamily);
00682 
00683                 this.FileName = resolved.FileName;
00684                 this.FamilyName = resolved.FamilyName;
00685                 this.TrueTypeFile = resolved.TrueTypeFile;
00686                 this.IsOblique = resolved.IsOblique;
00687                 this.IsBold = resolved.IsBold;
00688                 this.IsItalic = resolved.IsItalic;
00689                 this.IsStandardFamily = resolved.IsStandardFamily;
00690             }
00691         }
00692     }
00693 
00694 /// <summary>
00695 /// Represents a FontFamily created from a resource stream.
00696 /// </summary>
00697     public class ResourceFontFamily :  FontFamily
00698     {
00699 /// <summary>
00700 /// The name of the embedded resource, which will be parsed using
      <code>Avalonia.Media.FontFamily.Parse(string, Uri)</code>.
00701 /// </summary>
00702         public string ResourceName;
00703 
00704 /// <summary>
00705 /// Create a new <see cref="ResourceFontFamily"/> from the specified <paramref name="resourceStream"/>
      containing a TTF file, passing the specified <paramref name="resourceName"/> to the
      <code>Avalonia.Media.FontFamily.Parse(string, Uri)"</code> method.
00706 /// </summary>
00707 /// <param name="resourceStream">A resource stream containing a TTF file.</param>
00708 /// <param name="resourceName">The name of the embedded resource, which will be parsed using
      <code>Avalonia.Media.FontFamily.Parse(string, Uri)</code>.</param>
00709         public ResourceFontFamily(System.IO.Stream resourceStream, string resourceName) :
      base(resourceStream)
00710         {
00711             this.ResourceName = resourceName;
00712         }
00713     }
00714 
00715 }
```

8.80 FontLibrary.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
```

```
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021 using System.IO;
00022
00023 namespace VectSharp
00024 {
00025 /// <summary>
00026 /// Represents a font library with methods to create <see cref="FontFamily"/> objects from a string or
      from <see cref="FontFamily.StandardFontFamilies"/>.
00027 /// </summary>
00028     public interface IFontLibrary
00029     {
00030 /// <summary>
00031 /// Create a new font family from the specified family name or true type file.  If the family name or
      the true type file are not valid, an exception might be raised.
00032 /// </summary>
00033 /// <param name="fontFamily">The name of the font family to create, or the path to a TTF file.</param>
00034 /// <returns>If the font family name or the true type file is valid, a <see cref="FontFamily"/> object
      corresponding to the specified font family.</returns>
00035         FontFamily ResolveFontFamily(string fontFamily);
00036
00037 /// <summary>
00038 /// Create a new font family from the specified family name or true type file.  If the family name or
      the true type file are not valid, try to instantiate the font family using
00039 /// the <paramref name="fallback"/>.  If none of the fallback family names or true type files are
      valid, an exception might be raised.
00040 /// </summary>
00041 /// <param name="fontFamily">The name of the font family to create, or the path to a TTF file.</param>
00042 /// <param name="fallback">Names of additional font families or TTF files, which will be tried if the
      first <paramref name="fontFamily"/> is not valid.</param>
00043 /// <returns>A <see cref="FontFamily"/> object corresponding to the first of the specified font
      families that is valid.</returns>
00044         FontFamily ResolveFontFamily(string fontFamily, params string[] fallback);
00045
00046 /// <summary>
00047 /// Create a new font family from the specified standard font family name.
00048 /// </summary>
00049 /// <param name="standardFontFamily">The standard name of the font family.</param>
00050 /// <returns>A <see cref="FontFamily"/> object corresponding to the specified font family.</returns>
00051         FontFamily ResolveFontFamily(FontFamily.StandardFontFamilies standardFontFamily);
00052
00053 /// <summary>
00054 /// Create a new font family from the specified family name or true type file.  If the family name or
      the true type file are not valid, try to instantiate the font family using
00055 /// the <paramref name="fallback"/>.  If none of the fallback family names or true type files are
      valid, instantiate a standard font family using the <paramref name="finalFallback"/>.
00056 /// </summary>
00057 /// <param name="fontFamily">The name of the font family to create, or the path to a TTF file.</param>
00058 /// <param name="fallback">Names of additional font families or TTF files, which will be tried if the
      first <paramref name="fontFamily"/> is not valid.</param>
00059 /// <param name="finalFallback">The standard name of the font family that will be used if none of the
      fallback families are valid.</param>
00060 /// <returns>A <see cref="FontFamily"/> object corresponding to the first of the specified font
      families that is valid.</returns>
00061         FontFamily ResolveFontFamily(string fontFamily, FontFamily.StandardFontFamilies finalFallback,
      params string[] fallback);
00062     }
00063
00064 /// <summary>
00065 /// Abstract class with a default implementation of font family fallbacks.
00066 /// </summary>
00067     public abstract class FontLibrary :  IFontLibrary
00068     {
00069 /// <inheritdoc/>
00070         public abstract FontFamily ResolveFontFamily(string fontFamily);
00071
00072 /// <inheritdoc/>
00073         public abstract FontFamily ResolveFontFamily(FontFamily.StandardFontFamilies
      standardFontFamily);
00074
00075 /// <inheritdoc/>
00076         public virtual FontFamily ResolveFontFamily(string fontFamily, params string[] fallback)
00077         {
00078             bool found = false;
```

```
00079
00080             FontFamily tbr = null;
00081
00082             try
00083             {
00084                 tbr = ResolveFontFamily(fontFamily);
00085                 if (tbr.TrueTypeFile != null)
00086                 {
00087                     found = true;
00088                 }
00089             }
00090             catch
00091             {
00092                 tbr = null;
00093                 found = false;
00094             }
00095
00096             if (!found)
00097             {
00098                 for (int i = 0; i < fallback.Length; i++)
00099                 {
00100                     try
00101                     {
00102                         tbr = ResolveFontFamily(fontFamily);
00103                         if (tbr.TrueTypeFile != null)
00104                         {
00105                             found = true;
00106                         }
00107                     }
00108                     catch
00109                     {
00110                         tbr = null;
00111                         found = false;
00112                     }
00113
00114                     if (found)
00115                     {
00116                         return tbr;
00117                     }
00118                 }
00119             }
00120             else
00121             {
00122                 return tbr;
00123             }
00124
00125             throw new FontFamilyCreationException(fontFamily);
00126         }
00127
00128 /// <inheritdoc/>
00129         public virtual FontFamily ResolveFontFamily(string fontFamily, FontFamily.StandardFontFamilies
      finalFallback, params string[] fallback)
00130         {
00131             bool found = false;
00132
00133             FontFamily tbr = null;
00134
00135             try
00136             {
00137                 tbr = ResolveFontFamily(fontFamily);
00138                 if (tbr.TrueTypeFile != null)
00139                 {
00140                     found = true;
00141                 }
00142             }
00143             catch
00144             {
00145                 tbr = null;
00146                 found = false;
00147             }
00148
00149             if (!found)
00150             {
00151                 for (int i = 0; i < fallback.Length; i++)
00152                 {
00153                     try
00154                     {
00155                         tbr = ResolveFontFamily(fontFamily);
00156                         if (tbr.TrueTypeFile != null)
00157                         {
00158                             found = true;
00159                         }
00160                     }
00161                     catch
00162                     {
00163                         tbr = null;
00164                         found = false;
```

```
00165                     }
00166
00167                     if (found)
00168                     {
00169                         return tbr;
00170                     }
00171                 }
00172             }
00173             else
00174             {
00175                 return tbr;
00176             }
00177
00178             return ResolveFontFamily(finalFallback);
00179         }
00180     }
00181
00182 /// <summary>
00183 /// A font library that can be used to cache and resolve font family names.
00184 /// </summary>
00185     public class SimpleFontLibrary :  FontLibrary
00186     {
00187         private Dictionary<string, string> KnownFonts = new Dictionary<string, string>();
00188         private Dictionary<string, string> NotLoadedFonts = new Dictionary<string, string>();
00189         private Dictionary<string, FontFamily> LoadedFonts = new Dictionary<string, FontFamily>();
00190         private Dictionary<FontFamily.StandardFontFamilies, FontFamily> Fallbacks = new
    Dictionary<FontFamily.StandardFontFamilies, FontFamily>();
00191         private DefaultFontLibrary defaultLibrary = new DefaultFontLibrary();
00192
00193 /// <summary>
00194 /// Create a new <see cref="SimpleFontLibrary"/> instance.
00195 /// </summary>
00196 /// <param name="standardFontLibrary">An existing font library that will be used to resolve the
    standard font families.</param>
00197         public SimpleFontLibrary(IFontLibrary standardFontLibrary)
00198         {
00199             for (int i = 0; i < 14; i++)
00200             {
00201                 FontFamily.StandardFontFamilies stdFF = (FontFamily.StandardFontFamilies)i;
00202                 FontFamily resolved = standardFontLibrary.ResolveFontFamily(stdFF);
00203
00204                 Fallbacks[stdFF] = resolved;
00205                 LoadedFonts[FontFamily.StandardFamilies[i]] = resolved;
00206                 KnownFonts[FontFamily.StandardFamilies[i]] = FontFamily.StandardFamilies[i];
00207                 KnownFonts[resolved.FamilyName] = FontFamily.StandardFamilies[i];
00208                 KnownFonts[resolved.TrueTypeFile.GetFontFamilyName()] =
    FontFamily.StandardFamilies[i];
00209                 KnownFonts[resolved.TrueTypeFile.GetFullFontFamilyName()] =
    FontFamily.StandardFamilies[i];
00210                 KnownFonts[resolved.TrueTypeFile.GetFontName()] = FontFamily.StandardFamilies[i];
00211             }
00212         }
00213
00214 /// <summary>
00215 /// Create a new <see cref="SimpleFontLibrary"/> instance, using the default font library to resolve
    the standard font families.
00216 /// </summary>
00217         public SimpleFontLibrary() :  this(FontFamily.DefaultFontLibrary)
00218         {
00219
00220         }
00221
00222 /// <summary>
00223 /// Create a new <see cref="SimpleFontLibrary"/> instance, with the specified replacements for the
    standard font families.
00224 /// </summary>
00225 /// <param name="timesRoman">The font family to use for the Times-Roman standard font.</param>
00226 /// <param name="timesBold">The font family to use for the Times-Bold standard font.</param>
00227 /// <param name="timesItalic">The font family to use for the Times-Italic standard font.</param>
00228 /// <param name="timesBoldItalic">The font family to use for the Times-BoldItalic standard
    font.</param>
00229 /// <param name="helvetica">The font family to use for the Helvetica standard font.</param>
00230 /// <param name="helveticaBold">The font family to use for the Helvetica-Bold standard font.</param>
00231 /// <param name="helveticaOblique">The font family to use for the Helvetica-Oblique standard
    font.</param>
00232 /// <param name="helveticaBoldOblique">The font family to use for the Helvetica-BoldOblique standard
    font.</param>
00233 /// <param name="courier">The font family to use for the Courier standard font.</param>
00234 /// <param name="courierBold">The font family to use for the Courier-Bold standard font.</param>
00235 /// <param name="courierOblique">The font family to use for the Courier-Oblique standard font.</param>
00236 /// <param name="courierBoldOblique">The font family to use for the Courier-BoldOblique standard
    font.</param>
00237 /// <param name="symbol">The font family to use for the Symbol standard font.</param>
00238 /// <param name="zapfdingbats">The font family to use for the Zapfdingbats standard font.</param>
00239         public SimpleFontLibrary(FontFamily timesRoman, FontFamily timesBold, FontFamily timesItalic,
    FontFamily timesBoldItalic,
00240             FontFamily helvetica, FontFamily helveticaBold, FontFamily helveticaOblique, FontFamily
```

```
    helveticaBoldOblique,
00241             FontFamily courier, FontFamily courierBold, FontFamily courierOblique, FontFamily
    courierBoldOblique,
00242             FontFamily symbol, FontFamily zapfdingbats)
00243         {
00244             Fallbacks[FontFamily.StandardFontFamilies.TimesRoman] = timesRoman;
00245             Fallbacks[FontFamily.StandardFontFamilies.TimesBold] = timesBold;
00246             Fallbacks[FontFamily.StandardFontFamilies.TimesItalic] = timesItalic;
00247             Fallbacks[FontFamily.StandardFontFamilies.TimesBoldItalic] = timesBoldItalic;
00248 
00249             Fallbacks[FontFamily.StandardFontFamilies.Helvetica] = helvetica;
00250             Fallbacks[FontFamily.StandardFontFamilies.HelveticaBold] = helveticaBold;
00251             Fallbacks[FontFamily.StandardFontFamilies.HelveticaOblique] = helveticaOblique;
00252             Fallbacks[FontFamily.StandardFontFamilies.HelveticaBoldOblique] = helveticaBoldOblique;
00253 
00254             Fallbacks[FontFamily.StandardFontFamilies.Courier] = courier;
00255             Fallbacks[FontFamily.StandardFontFamilies.CourierBold] = courierBold;
00256             Fallbacks[FontFamily.StandardFontFamilies.CourierOblique] = courierOblique;
00257             Fallbacks[FontFamily.StandardFontFamilies.CourierBoldOblique] = courierBoldOblique;
00258 
00259             Fallbacks[FontFamily.StandardFontFamilies.Symbol] = symbol;
00260             Fallbacks[FontFamily.StandardFontFamilies.ZapfDingbats] = zapfdingbats;
00261 
00262             for (int i = 0; i < 14; i++)
00263             {
00264                 FontFamily.StandardFontFamilies stdFF = (FontFamily.StandardFontFamilies)i;
00265                 FontFamily resolved = Fallbacks[stdFF];
00266                 resolved.IsStandardFamily = true;
00267                 resolved.FileName = FontFamily.StandardFamilies[i];
00268                 resolved.FamilyName = FontFamily.StandardFamilies[i].Replace("-", " ");
00269 
00270                 LoadedFonts[FontFamily.StandardFamilies[i]] = resolved;
00271                 KnownFonts[FontFamily.StandardFamilies[i]] = FontFamily.StandardFamilies[i];
00272                 KnownFonts[resolved.FamilyName] = FontFamily.StandardFamilies[i];
00273                 KnownFonts[resolved.TrueTypeFile.GetFontFamilyName()] =
    FontFamily.StandardFamilies[i];
00274                 KnownFonts[resolved.TrueTypeFile.GetFullFontFamilyName()] =
    FontFamily.StandardFamilies[i];
00275                 KnownFonts[resolved.TrueTypeFile.GetFontName()] = FontFamily.StandardFamilies[i];
00276             }
00277         }
00278 
00279 
00280 /// <summary>
00281 /// Create a new <see cref="SimpleFontLibrary"/> instance, with the specified replacements for the
    standard font families.
00282 /// </summary>
00283 /// <param name="timesRoman">The font family to use for the Times-Roman standard font.</param>
00284 /// <param name="timesBold">The font family to use for the Times-Bold standard font.</param>
00285 /// <param name="timesItalic">The font family to use for the Times-Italic standard font.</param>
00286 /// <param name="timesBoldItalic">The font family to use for the Times-BoldItalic standard
    font.</param>
00287 /// <param name="helvetica">The font family to use for the Helvetica standard font.</param>
00288 /// <param name="helveticaBold">The font family to use for the Helvetica-Bold standard font.</param>
00289 /// <param name="helveticaOblique">The font family to use for the Helvetica-Oblique standard
    font.</param>
00290 /// <param name="helveticaBoldOblique">The font family to use for the Helvetica-BoldOblique standard
    font.</param>
00291 /// <param name="courier">The font family to use for the Courier standard font.</param>
00292 /// <param name="courierBold">The font family to use for the Courier-Bold standard font.</param>
00293 /// <param name="courierOblique">The font family to use for the Courier-Oblique standard font.</param>
00294 /// <param name="courierBoldOblique">The font family to use for the Courier-BoldOblique standard
    font.</param>
00295 /// <param name="symbol">The font family to use for the Symbol standard font.</param>
00296 /// <param name="zapfdingbats">The font family to use for the Zapfdingbats standard font.</param>
00297         public SimpleFontLibrary(string timesRoman, string timesBold, string timesItalic, string
    timesBoldItalic,
00298             string helvetica, string helveticaBold, string helveticaOblique, string
    helveticaBoldOblique,
00299             string courier, string courierBold, string courierOblique, string courierBoldOblique,
00300             string symbol, string zapfdingbats)
00301         {
00302             Fallbacks[FontFamily.StandardFontFamilies.TimesRoman] =
    FontFamily.DefaultFontLibrary.ResolveFontFamily(timesRoman);
00303             Fallbacks[FontFamily.StandardFontFamilies.TimesBold] =
    FontFamily.DefaultFontLibrary.ResolveFontFamily(timesBold);
00304             Fallbacks[FontFamily.StandardFontFamilies.TimesItalic] =
    FontFamily.DefaultFontLibrary.ResolveFontFamily(timesItalic);
00305             Fallbacks[FontFamily.StandardFontFamilies.TimesBoldItalic] =
    FontFamily.DefaultFontLibrary.ResolveFontFamily(timesBoldItalic);
00306 
00307             Fallbacks[FontFamily.StandardFontFamilies.Helvetica] =
    FontFamily.DefaultFontLibrary.ResolveFontFamily(helvetica);
00308             Fallbacks[FontFamily.StandardFontFamilies.HelveticaBold] =
    FontFamily.DefaultFontLibrary.ResolveFontFamily(helveticaBold);
00309             Fallbacks[FontFamily.StandardFontFamilies.HelveticaOblique] =
    FontFamily.DefaultFontLibrary.ResolveFontFamily(helveticaOblique);
```

```
00310             Fallbacks[FontFamily.StandardFontFamilies.HelveticaBoldOblique] =
      FontFamily.DefaultFontLibrary.ResolveFontFamily(helveticaBoldOblique);
00311
00312             Fallbacks[FontFamily.StandardFontFamilies.Courier] =
      FontFamily.DefaultFontLibrary.ResolveFontFamily(courier);
00313             Fallbacks[FontFamily.StandardFontFamilies.CourierBold] =
      FontFamily.DefaultFontLibrary.ResolveFontFamily(courierBold);
00314             Fallbacks[FontFamily.StandardFontFamilies.CourierOblique] =
      FontFamily.DefaultFontLibrary.ResolveFontFamily(courierOblique);
00315             Fallbacks[FontFamily.StandardFontFamilies.CourierBoldOblique] =
      FontFamily.DefaultFontLibrary.ResolveFontFamily(courierBoldOblique);
00316
00317             Fallbacks[FontFamily.StandardFontFamilies.Symbol] =
      FontFamily.DefaultFontLibrary.ResolveFontFamily(symbol);
00318             Fallbacks[FontFamily.StandardFontFamilies.ZapfDingbats] =
      FontFamily.DefaultFontLibrary.ResolveFontFamily(zapfdingbats);
00319
00320             for (int i = 0; i < 14; i++)
00321             {
00322                 FontFamily.StandardFontFamilies stdFF = (FontFamily.StandardFontFamilies)i;
00323                 FontFamily resolved = Fallbacks[stdFF];
00324                 resolved.IsStandardFamily = true;
00325                 resolved.FileName = FontFamily.StandardFamilies[i];
00326                 resolved.FamilyName = FontFamily.StandardFamilies[i].Replace("-", " ");
00327
00328                 LoadedFonts[FontFamily.StandardFamilies[i]] = resolved;
00329                 KnownFonts[FontFamily.StandardFamilies[i]] = FontFamily.StandardFamilies[i];
00330                 KnownFonts[resolved.FamilyName] = FontFamily.StandardFamilies[i];
00331                 KnownFonts[resolved.TrueTypeFile.GetFontFamilyName()] =
      FontFamily.StandardFamilies[i];
00332                 KnownFonts[resolved.TrueTypeFile.GetFullFontFamilyName()] =
      FontFamily.StandardFamilies[i];
00333                 KnownFonts[resolved.TrueTypeFile.GetFontName()] = FontFamily.StandardFamilies[i];
00334             }
00335         }
00336
00337
00338 /// <summary>
00339 /// Add the specified font family to the library with the specified name.
00340 /// </summary>
00341 /// <param name="fontFamilyName">The name of the font family.</param>
00342 /// <param name="fontFamily">The font family to add.</param>
00343         public void Add(string fontFamilyName, FontFamily fontFamily)
00344         {
00345             this.LoadedFonts[fontFamilyName] = fontFamily;
00346             this.KnownFonts[fontFamilyName] = fontFamilyName;
00347
00348             if (fontFamily.TrueTypeFile != null)
00349             {
00350                 this.KnownFonts[fontFamily.TrueTypeFile.GetFontFamilyName()] = fontFamilyName;
00351                 this.KnownFonts[fontFamily.TrueTypeFile.GetFullFontFamilyName()] = fontFamilyName;
00352                 this.KnownFonts[fontFamily.TrueTypeFile.GetFontName()] = fontFamilyName;
00353             }
00354         }
00355
00356 /// <summary>
00357 /// Add the specified font family to the library.
00358 /// </summary>
00359 /// <param name="fontFamily">The font family to add.</param>
00360         public void Add(FontFamily fontFamily)
00361         {
00362             if (fontFamily.TrueTypeFile != null)
00363             {
00364                 this.Add(fontFamily.TrueTypeFile.GetFullFontFamilyName(), fontFamily);
00365             }
00366         }
00367
00368 /// <summary>
00369 /// Add the font family contained in the specified True Type Font file to the library.
00370 /// </summary>
00371 /// <param name="fileName">The path to the TTF file containing the font family.</param>
00372         public void Add(string fileName)
00373         {
00374             FontFamily fontFamily = FontFamily.DefaultFontLibrary.ResolveFontFamily(fileName);
00375
00376             if (fontFamily.TrueTypeFile != null)
00377             {
00378                 this.Add(fontFamily.TrueTypeFile.GetFullFontFamilyName(), fontFamily);
00379             }
00380         }
00381
00382 /// <summary>
00383 /// Add the font family contained in the specified True Type Font file to the library, with the
      specified name.  The font family is not loaded until it is requested for the first time.
00384 /// </summary>
00385 /// <param name="fontFamily">The name of the font family.</param>
00386 /// <param name="fileName">The path to the TTF file containing the font family.</param>
```

```
00387         public void Add(string fontFamily, string fileName)
00388         {
00389             this.KnownFonts[fontFamily] = fontFamily;
00390             this.NotLoadedFonts[fontFamily] = fileName;
00391         }
00392 
00393 /// <inheritdoc/>
00394         public override FontFamily ResolveFontFamily(FontFamily.StandardFontFamilies
      standardFontFamily)
00395         {
00396             return Fallbacks[standardFontFamily];
00397         }
00398 
00399 /// <inheritdoc/>
00400         public override FontFamily ResolveFontFamily(string fontFamily)
00401         {
00402             if (KnownFonts.TryGetValue(fontFamily, out string knownFontName))
00403             {
00404                 if (LoadedFonts.TryGetValue(knownFontName, out FontFamily tbr))
00405                 {
00406                     return tbr;
00407                 }
00408                 else
00409                 {
00410                     if (NotLoadedFonts.TryGetValue(knownFontName, out string ttfFile))
00411                     {
00412                         tbr = defaultLibrary.ResolveFontFamily(ttfFile);
00413 
00414                         if (tbr.TrueTypeFile != null)
00415                         {
00416                             string familyName = tbr.TrueTypeFile.GetFontFamilyName();
00417 
00418                             this.LoadedFonts[familyName] = tbr;
00419                             this.KnownFonts[familyName] = familyName;
00420                             this.KnownFonts[tbr.TrueTypeFile.GetFontName()] = familyName;
00421                         }
00422 
00423                         return tbr;
00424                     }
00425                     else
00426                     {
00427                         return defaultLibrary.ResolveFontFamily(fontFamily);
00428                     }
00429                 }
00430             }
00431             else
00432             {
00433                 return defaultLibrary.ResolveFontFamily(fontFamily);
00434             }
00435         }
00436     }
00437 
00438 /// <summary>
00439 /// An exception that occurs while creating a <see cref="FontFamily"/>.
00440 /// </summary>
00441     public class FontFamilyCreationException :  Exception
00442     {
00443 /// <summary>
00444 /// The name of the font family that was being created.
00445 /// </summary>
00446         public string FontFamily { get; }
00447 
00448 /// <summary>
00449 /// Create a new <see cref="FontFamilyCreationException"/> instance.
00450 /// </summary>
00451 /// <param name="fontFamily">The name of the font family that was being created.</param>
00452         public FontFamilyCreationException(string fontFamily) :  base("The font family \"" +
      fontFamily + "\" could not be created!")
00453         {
00454             this.FontFamily = fontFamily;
00455         }
00456     }
00457 
00458 /// <summary>
00459 /// A default font library that resolves standard families using the embedded fonts.
00460 /// </summary>
00461     public class DefaultFontLibrary :  FontLibrary
00462     {
00463 /// <inheritdoc/>
00464         public override FontFamily ResolveFontFamily(string fontFamily)
00465         {
00466             lock (FontFamily.fontFamilyLock)
00467             {
00468                 bool isStandardFamily;
00469 
00470                 if (FontFamily.StandardFamilies.Contains(fontFamily) ||
      FontFamily.StandardFamilies.Contains(fontFamily.Replace(" ", "-")))
```

```
00471                 {
00472                     isStandardFamily = true;
00473                 }
00474                 else
00475                 {
00476                     isStandardFamily = false;
00477                 }
00478 
00479                 if (isStandardFamily)
00480                 {
00481                     fontFamily = fontFamily.Replace(" ", "-");
00482                     Stream ttfStream =
      FontFamily.GetManifestResourceStream(FontFamily.StandardFontFamilyResources[Array.IndexOf(FontFamily.StandardFamilies,
      fontFamily)]);
00483 
00484                     FontFamily tbr = new FontFamily(ttfStream);
00485                     tbr.IsStandardFamily = true;
00486                     tbr.FileName = fontFamily;
00487                     tbr.FamilyName = tbr.FileName.Replace("-", " ");
00488 
00489                     if (fontFamily == "Times-Italic" || fontFamily == "Times-BoldItalic" || fontFamily
      == "Helvetica-Oblique" || fontFamily == "Helvetica-BoldOblique" || fontFamily == "Courier-Oblique" ||
      fontFamily == "Courier-BoldOblique")
00490                     {
00491                         tbr.IsItalic = true;
00492                         tbr.IsOblique = (fontFamily == "Courier-Oblique" || fontFamily ==
      "Courier-BoldOblique");
00493                     }
00494                     else
00495                     {
00496                         tbr.IsItalic = false;
00497                         tbr.IsOblique = false;
00498                     }
00499 
00500                     return tbr;
00501                 }
00502                 else
00503                 {
00504                     try
00505                     {
00506                         FontFamily tbr = new FontFamily(TrueTypeFile.CreateTrueTypeFile(fontFamily));
00507                         tbr.FileName = fontFamily;
00508                         tbr.FamilyName = tbr.TrueTypeFile?.GetFullFontFamilyName() ??  tbr.FileName;
00509                         return tbr;
00510                     }
00511                     catch
00512                     {
00513                         FontFamily tbr = new FontFamily();
00514                         tbr.FileName = fontFamily;
00515                         tbr.FamilyName = fontFamily;
00516                         return tbr;
00517                     }
00518                 }
00519             }
00520         }
00521 
00522 /// <inheritdoc/>
00523         public override FontFamily ResolveFontFamily(FontFamily.StandardFontFamilies
      standardFontFamily)
00524         {
00525             lock (FontFamily.fontFamilyLock)
00526             {
00527                 Stream ttfStream =
      FontFamily.GetManifestResourceStream(FontFamily.StandardFontFamilyResources[(int)standardFontFamily]);
00528                 FontFamily tbr = new FontFamily(ttfStream);
00529 
00530 
00531                 tbr.IsStandardFamily = true;
00532 
00533                 tbr.FileName = FontFamily.StandardFamilies[(int)standardFontFamily];
00534                 tbr.FamilyName = tbr.FileName.Replace("-", " ");
00535 
00536                 if (tbr.FileName == "Times-Italic" || tbr.FileName == "Times-BoldItalic" ||
      tbr.FileName == "Helvetica-Oblique" || tbr.FileName == "Helvetica-BoldOblique" || tbr.FileName ==
      "Courier-Oblique" || tbr.FileName == "Courier-BoldOblique")
00537                 {
00538                     tbr.IsItalic = true;
00539                     tbr.IsOblique = (tbr.FileName == "Courier-Oblique" || tbr.FileName ==
      "Courier-BoldOblique");
00540                 }
00541                 else
00542                 {
00543                     tbr.IsItalic = false;
00544                     tbr.IsOblique = false;
00545                 }
00546 
00547                 return tbr;
```

```
00548             }
00549         }
00550     }
00551
00552 /// <summary>
00553 /// A font library that resolves fonts from a folder containing TrueType files.
00554 /// </summary>
00555     public class FolderFontLibrary :  FontLibrary
00556     {
00557         private Dictionary<string, FontFamily> LoadedFonts = new Dictionary<string, FontFamily>();
00558         private Dictionary<string, string> KnownFonts = new Dictionary<string, string>();
00559         private IFontLibrary StandardFontLibrary;
00560
00561 /// <summary>
00562 /// Creates a new <see cref="FolderFontLibrary"/> using fonts from the specified path, and using the
    <see cref="FontFamily.DefaultFontLibrary"/> to resolve the standard font families.
00563 /// </summary>
00564 /// <param name="folderPath">The path to the folder containing the TrueType files.</param>
00565         public FolderFontLibrary(string folderPath) :  this(folderPath, FontFamily.DefaultFontLibrary)
    { }
00566
00567 /// <summary>
00568 /// Creates a new <see cref="FolderFontLibrary"/> using fonts from the specified path, and using the
    specified <see cref="IFontLibrary"/> to resolve the standard font families.
00569 /// </summary>
00570 /// <param name="folderPath">The path to the folder containing the TrueType files.</param>
00571 /// <param name="standardFontLibrary">The <see cref="IFontLibrary"/> to use when resolving standard
    font families.</param>
00572         public FolderFontLibrary(string folderPath, IFontLibrary standardFontLibrary)
00573         {
00574             foreach (string file in Directory.GetFiles(folderPath))
00575             {
00576                 bool isValid = TrueTypeFile.GetNames(file, out List<TrueTypeFile.TrueTypeName> names);
00577
00578                 if (isValid && names != null)
00579                 {
00580                     foreach (TrueTypeFile.TrueTypeName name in names)
00581                     {
00582                         switch (name.NameId)
00583                         {
00584                             case TrueTypeFile.TrueTypeName.NameIdentifier.FullName:
00585                             case TrueTypeFile.TrueTypeName.NameIdentifier.PostScriptName:
00586                                 KnownFonts[name.Name] = file;
00587                                 break;
00588                             case TrueTypeFile.TrueTypeName.NameIdentifier.FontFamily:
00589                             case TrueTypeFile.TrueTypeName.NameIdentifier.PreferredFamily:
00590                                 if (!KnownFonts.ContainsKey(name.Name))
00591                                 {
00592                                     KnownFonts[name.Name] = file;
00593                                 }
00594                                 break;
00595                         }
00596                     }
00597                 }
00598             }
00599
00600             StandardFontLibrary = standardFontLibrary;
00601         }
00602
00603 /// <inheritdoc/>
00604         public override FontFamily ResolveFontFamily(string fontFamily)
00605         {
00606             if (KnownFonts.TryGetValue(fontFamily, out string knownFontName))
00607             {
00608                 if (LoadedFonts.TryGetValue(knownFontName, out FontFamily tbr))
00609                 {
00610                     return tbr;
00611                 }
00612                 else
00613                 {
00614                     tbr = new FontFamily(TrueTypeFile.CreateTrueTypeFile(knownFontName));
00615                     tbr.FileName = knownFontName;
00616                     tbr.FamilyName = tbr.TrueTypeFile?.GetFullFontFamilyName() ??  tbr.FileName;
00617
00618                     if (tbr.TrueTypeFile != null)
00619                     {
00620                         this.LoadedFonts[knownFontName] = tbr;
00621                     }
00622
00623                     return tbr;
00624                 }
00625             }
00626             else
00627             {
00628                 FontFamily tbr = new FontFamily();
00629                 tbr.FileName = fontFamily;
00630                 tbr.FamilyName = fontFamily;
```

```
00631                 return tbr;
00632             }
00633         }
00634 
00635 /// <inheritdoc/>
00636         public override FontFamily ResolveFontFamily(FontFamily.StandardFontFamilies
      standardFontFamily)
00637         {
00638             FontFamily attempt =
      ResolveFontFamily(FontFamily.StandardFamilies[(int)standardFontFamily]);
00639 
00640             if (attempt != null && attempt.TrueTypeFile != null)
00641             {
00642                 return attempt;
00643             }
00644             else
00645             {
00646                 return StandardFontLibrary.ResolveFontFamily(standardFontFamily);
00647             }
00648         }
00649     }
00650 
00651 /// <summary>
00652 /// A font library that tries to resolve fonts using other font libraries.
00653 /// </summary>
00654     public class MultiFontLibrary :  FontLibrary
00655     {
00656         private FontLibrary[] Libraries;
00657         private DefaultFontLibrary DefaultLibrary = new DefaultFontLibrary();
00658 
00659 /// <summary>
00660 /// Creates a new <see cref="MultiFontLibrary"/> resolving fonts using the specified <paramref
      name="libraries"/>.
00661 /// </summary>
00662 /// <param name="libraries">The font libraries that will be used, in order, to resolve the font
      families.</param>
00663 /// <exception cref="ArgumentException">Thrown if the <paramref name="libraries"/> do not contain any
      element.</exception>
00664         public MultiFontLibrary(params FontLibrary[] libraries) :
      this((IEnumerable<FontLibrary>)libraries) { }
00665 
00666 
00667 /// <summary>
00668 /// Creates a new <see cref="MultiFontLibrary"/> resolving fonts using the specified <paramref
      name="libraries"/>.
00669 /// </summary>
00670 /// <param name="libraries">The font libraries that will be used, in order, to resolve the font
      families.</param>
00671 /// <exception cref="ArgumentException">Thrown if the <paramref name="libraries"/> do not contain any
      element.</exception>
00672         public MultiFontLibrary(IEnumerable<FontLibrary> libraries)
00673         {
00674             this.Libraries = libraries.ToArray();
00675 
00676             if (this.Libraries.Length == 0)
00677             {
00678                 throw new ArgumentException("No font library has been provided!");
00679             }
00680         }
00681 
00682 /// <inheritdoc/>
00683         public override FontFamily ResolveFontFamily(string fontFamily)
00684         {
00685             for (int i = 0; i < Libraries.Length; i++)
00686             {
00687                 FontFamily attempt = Libraries[i].ResolveFontFamily(fontFamily);
00688 
00689                 if (attempt != null && attempt.TrueTypeFile != null)
00690                 {
00691                     return attempt;
00692                 }
00693             }
00694 
00695             return DefaultLibrary.ResolveFontFamily(fontFamily);
00696         }
00697 
00698 /// <inheritdoc/>
00699         public override FontFamily ResolveFontFamily(FontFamily.StandardFontFamilies
      standardFontFamily)
00700         {
00701             for (int i = 0; i < Libraries.Length; i++)
00702             {
00703                 FontFamily attempt = Libraries[i].ResolveFontFamily(standardFontFamily);
00704 
00705                 if (attempt != null && attempt.TrueTypeFile != null)
00706                 {
00707                     return attempt;
```

```
00708                 }
00709             }
00710
00711             return DefaultLibrary.ResolveFontFamily(standardFontFamily);
00712         }
00713     }
00714
00715 }
```

8.81 FormattedText.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Text;
00021 using System.Linq;
00022 using System.Diagnostics;
00023
00024 namespace VectSharp
00025 {
00026 /// <summary>
00027 /// Represents the position of the text.
00028 /// </summary>
00029     public enum Script
00030     {
00031 /// <summary>
00032 /// The text is normal text.
00033 /// </summary>
00034         Normal,
00035
00036 /// <summary>
00037 /// The text is a superscript.
00038 /// </summary>
00039         Superscript,
00040
00041 /// <summary>
00042 /// The text is a subscript.
00043 /// </summary>
00044         Subscript
00045     }
00046
00047 /// <summary>
00048 /// Represents a run of text that should be drawn with the same style.
00049 /// </summary>
00050     [DebuggerDisplay("{Text}")]
00051     public class FormattedText
00052     {
00053 /// <summary>
00054 /// Represents the text represented by this instance.
00055 /// </summary>
00056         public string Text { get; }
00057
00058 /// <summary>
00059 /// Represents the font that should be used to draw the text.
00060 /// </summary>
00061         public Font Font { get; }
00062
00063 /// <summary>
00064 /// Represents the position of the text.
00065 /// </summary>
00066         public Script Script { get; }
00067
00068 /// <summary>
00069 /// Represents the brush that should be used to draw the text.  If this is null, the default brush is
      used.
00070 /// </summary>
00071         public Brush Brush { get; }
```

```
00072
00073 /// <summary>
00074 /// Represents the text spacing parameters used to draw the text.
00075 /// </summary>
00076         public TextSpacing Spacing { get; }
00077
00078 /// <summary>
00079 /// Creates a new <see cref="FormattedText"/> instance with the specified <paramref name="text"/>,
      <paramref name="font"/>, <paramref name="script"/> position and <paramref name="brush"/>.
00080 /// </summary>
00081 /// <param name="text">The text that will be contained in the new <see cref="FormattedText"/>.</param>
00082 /// <param name="font">The font that will be used by the new <see cref="FormattedText"/>.</param>
00083 /// <param name="script">The script position of the new <see cref="FormattedText"/>.</param>
00084 /// <param name="brush">The brush that will be used by the new <see cref="FormattedText"/>.</param>
00085 /// <param name="spacing">The text spacing parameters used to draw the text.</param>
00086         public FormattedText(string text, Font font, Script script = Script.Normal, Brush brush =
      null, TextSpacing?  spacing = null)
00087         {
00088             this.Text = text ??  "";
00089             this.Font = font;
00090             this.Script = script;
00091             this.Brush = brush;
00092             this.Spacing = spacing ??  TextSpacing.Default;
00093         }
00094
00095 /// <summary>
00096 /// Parse the formatting information contained in a text string into a collection of <see
      cref="FormattedText"/> objects.
00097 /// </summary>
00098 /// <param name="text">
00099 /// The string containing formatting information.  Format information is specified using HTML-like
      tags:
00100 /// <list type="bullet">
00101 /// <item><c>&lt;b&gt;&lt;/b&gt;</c> or <c>&lt;strong&gt;&lt;/strong&gt;</c> are used for bold
      text;</item>
00102 /// <item><c>&lt;i&gt;&lt;/i&gt;</c> or <c>&lt;em&gt;&lt;/em&gt;</c> are used for text in
      italics;</item>
00103 /// <item><c>&lt;u&gt;&lt;/u&gt;</c></item> are used for underlined text;
00104 /// <item><c>&lt;sup&gt;&lt;/sup&gt;</c> and <c>&lt;sub&gt;&lt;/sub&gt;</c> are used, respectively,
      for superscript and subscript text;</item>
00105 /// <item><c>&lt;#COLOUR&gt;&lt;/#&gt;</c> is used to specify the colour of the text, where
      <c>COLOUR</c> is a CSS colour string (e.g.  <c>&lt;#red&gt;</c>, <c>&lt;#0080FF&gt;</c>, or
      <c>&lt;#rgba(128, 80, 52, 0.5)&gt;</c>).</item>
00106 /// </list></param>
00107 /// <param name="normalFont">The font that will be used for text that is neither bold nor
      italic.</param>
00108 /// <param name="boldFont">The font that will be used for text that is bold.  Note that this does not
      necessarily have to be a bold font; this is just the font that is applied to text contained within
      <c>&lt;b&gt;&lt;/b&gt;</c> tags.</param>
00109 /// <param name="italicFont">The font that will be used for text that is in italics.  Note that this
      does not necessarily have to be an italic font; this is just the font that is applied to text
      contained within <c>&lt;i&gt;&lt;/i&gt;</c> tags.</param>
00110 /// <param name="boldItalicFont">The font that will be used for text that is both in bold and in
      italics.</param>
00111 /// <param name="defaultBrush">The default <see cref="Brush"/> that will be used for text runs that do
      not specify a colour.  If this is <see langword="null"/>, the default <see cref="Brush"/> will be the
      one specified in the painting call.</param>
00112 /// <param name="spacing">The text spacing parameters used to draw the text.</param>
00113 /// <returns>A lazy collection of <see cref="FormattedText"/> objects.  Note that every enumeration of
      this collection causes the text to be parsed again; if you need to enumerate this collection more than
      once, you should probably convert it e.g.  to a <see cref="List{T}"/>.</returns>
00114         public static IEnumerable<FormattedText> Format(string text, Font normalFont, Font boldFont,
      Font italicFont, Font boldItalicFont, Brush defaultBrush = null, TextSpacing?  spacing = null)
00115         {
00116             if (string.IsNullOrEmpty(text))
00117             {
00118                 yield break;
00119             }
00120
00121             StringBuilder currentRun = new StringBuilder();
00122             int boldDepth = 0;
00123             int italicsDepth = 0;
00124             int underlineDepth = 0;
00125             int superscriptDepth = 0;
00126             int subscriptDepth = 0;
00127             Stack<Brush> brushes = new Stack<Brush>();
00128             brushes.Push(defaultBrush);
00129
00130             for (int i = 0; i < text.Length; i++)
00131             {
00132                 if (text[i] != '<')
00133                 {
00134                     currentRun.Append(text[i]);
00135                 }
00136                 else
00137                 {
00138                     Tags tag = GetTag(text, i, out int tagEnd, out Brush tagBrush);
```

```
00139
00140                   if (tag == Tags.None)
00141                   {
00142                       currentRun.Append(text[i]);
00143                   }
00144                   else
00145                   {
00146                       i = tagEnd;
00147
00148                       string txt = currentRun.ToString();
00149
00150                       switch (tag)
00151                       {
00152                           case Tags.BoldOpen:
00153                               if (!string.IsNullOrEmpty(txt))
00154                               {
00155                                   yield return new FormattedText(txt, GetFont(boldDepth % 2 == 1,
      italicsDepth % 2 == 1, underlineDepth % 2 == 1, normalFont, boldFont, italicFont, boldItalicFont),
      superscriptDepth > subscriptDepth ?  Script.Superscript :  subscriptDepth > superscriptDepth ?
      Script.Subscript :  Script.Normal, brushes.Peek(), spacing);
00156                               }
00157                               currentRun.Clear();
00158                               boldDepth++;
00159                               break;
00160                           case Tags.BoldClose:
00161                               if (boldDepth > 0)
00162                               {
00163                                   if (!string.IsNullOrEmpty(txt))
00164                                   {
00165                                       yield return new FormattedText(txt, GetFont(boldDepth % 2 ==
      1, italicsDepth % 2 == 1, underlineDepth % 2 == 1, normalFont, boldFont, italicFont, boldItalicFont),
      superscriptDepth > subscriptDepth ?  Script.Superscript :  subscriptDepth > superscriptDepth ?
      Script.Subscript :  Script.Normal, brushes.Peek(), spacing);
00166                                   }
00167                                   currentRun.Clear();
00168                                   boldDepth--;
00169                               }
00170                               break;
00171                           case Tags.UnderlineOpen:
00172                               if (!string.IsNullOrEmpty(txt))
00173                               {
00174                                   yield return new FormattedText(txt, GetFont(boldDepth % 2 == 1,
      italicsDepth % 2 == 1, underlineDepth % 2 == 1, normalFont, boldFont, italicFont, boldItalicFont),
      superscriptDepth > subscriptDepth ?  Script.Superscript :  subscriptDepth > superscriptDepth ?
      Script.Subscript :  Script.Normal, brushes.Peek(), spacing);
00175                               }
00176                               currentRun.Clear();
00177                               underlineDepth++;
00178                               break;
00179                           case Tags.UnderlineClose:
00180                               if (underlineDepth > 0)
00181                               {
00182                                   if (!string.IsNullOrEmpty(txt))
00183                                   {
00184                                       yield return new FormattedText(txt, GetFont(boldDepth % 2 ==
      1, italicsDepth % 2 == 1, underlineDepth % 2 == 1, normalFont, boldFont, italicFont, boldItalicFont),
      superscriptDepth > subscriptDepth ?  Script.Superscript :  subscriptDepth > superscriptDepth ?
      Script.Subscript :  Script.Normal, brushes.Peek(), spacing);
00185                                   }
00186                                   currentRun.Clear();
00187                                   underlineDepth--;
00188                               }
00189                               break;
00190
00191                           case Tags.ItalicsOpen:
00192                               if (!string.IsNullOrEmpty(txt))
00193                               {
00194                                   yield return new FormattedText(txt, GetFont(boldDepth % 2 == 1,
      italicsDepth % 2 == 1, underlineDepth % 2 == 1, normalFont, boldFont, italicFont, boldItalicFont),
      superscriptDepth > subscriptDepth ?  Script.Superscript :  subscriptDepth > superscriptDepth ?
      Script.Subscript :  Script.Normal, brushes.Peek(), spacing);
00195                               }
00196                               currentRun.Clear();
00197                               italicsDepth++;
00198                               break;
00199                           case Tags.ItalicsClose:
00200                               if (italicsDepth > 0)
00201                               {
00202                                   if (!string.IsNullOrEmpty(txt))
00203                                   {
00204                                       yield return new FormattedText(txt, GetFont(boldDepth % 2 ==
      1, italicsDepth % 2 == 1, underlineDepth % 2 == 1, normalFont, boldFont, italicFont, boldItalicFont),
      superscriptDepth > subscriptDepth ?  Script.Superscript :  subscriptDepth > superscriptDepth ?
      Script.Subscript :  Script.Normal, brushes.Peek(), spacing);
00205                                   }
00206                                   currentRun.Clear();
00207                                   italicsDepth--;
```

```
00208                                 }
00209                                 break;
00210
00211                             case Tags.SupOpen:
00212                                 if (!string.IsNullOrEmpty(txt))
00213                                 {
00214                                     yield return new FormattedText(txt, GetFont(boldDepth % 2 == 1,
      italicsDepth % 2 == 1, underlineDepth % 2 == 1, normalFont, boldFont, italicFont, boldItalicFont),
      superscriptDepth > subscriptDepth ?  Script.Superscript :  subscriptDepth > superscriptDepth ?
      Script.Subscript :  Script.Normal, brushes.Peek(), spacing);
00215                                 }
00216                                 currentRun.Clear();
00217                                 superscriptDepth++;
00218                                 break;
00219                             case Tags.SupClose:
00220                                 if (superscriptDepth > 0)
00221                                 {
00222                                     if (!string.IsNullOrEmpty(txt))
00223                                     {
00224                                         yield return new FormattedText(txt, GetFont(boldDepth % 2 ==
      1, italicsDepth % 2 == 1, underlineDepth % 2 == 1, normalFont, boldFont, italicFont, boldItalicFont),
      superscriptDepth > subscriptDepth ?  Script.Superscript :  subscriptDepth > superscriptDepth ?
      Script.Subscript :  Script.Normal, brushes.Peek(), spacing);
00225                                     }
00226                                     currentRun.Clear();
00227                                     superscriptDepth--;
00228                                 }
00229                                 break;
00230
00231                             case Tags.SubOpen:
00232                                 if (!string.IsNullOrEmpty(txt))
00233                                 {
00234                                     yield return new FormattedText(txt, GetFont(boldDepth % 2 == 1,
      italicsDepth % 2 == 1, underlineDepth % 2 == 1, normalFont, boldFont, italicFont, boldItalicFont),
      superscriptDepth > subscriptDepth ?  Script.Superscript :  subscriptDepth > superscriptDepth ?
      Script.Subscript :  Script.Normal, brushes.Peek(), spacing);
00235                                 }
00236                                 currentRun.Clear();
00237                                 subscriptDepth++;
00238                                 break;
00239                             case Tags.SubClose:
00240                                 if (subscriptDepth > 0)
00241                                 {
00242                                     if (!string.IsNullOrEmpty(txt))
00243                                     {
00244                                         yield return new FormattedText(txt, GetFont(boldDepth % 2 ==
      1, italicsDepth % 2 == 1, underlineDepth % 2 == 1, normalFont, boldFont, italicFont, boldItalicFont),
      superscriptDepth > subscriptDepth ?  Script.Superscript :  subscriptDepth > superscriptDepth ?
      Script.Subscript :  Script.Normal, brushes.Peek(), spacing);
00245                                     }
00246                                     currentRun.Clear();
00247                                     subscriptDepth--;
00248                                 }
00249                                 break;
00250                             case Tags.ColourOpen:
00251                                 if (!string.IsNullOrEmpty(txt))
00252                                 {
00253                                     yield return new FormattedText(txt, GetFont(boldDepth % 2 == 1,
      italicsDepth % 2 == 1, underlineDepth % 2 == 1, normalFont, boldFont, italicFont, boldItalicFont),
      superscriptDepth > subscriptDepth ?  Script.Superscript :  subscriptDepth > superscriptDepth ?
      Script.Subscript :  Script.Normal, brushes.Peek(), spacing);
00254                                 }
00255                                 currentRun.Clear();
00256                                 brushes.Push(tagBrush);
00257                                 break;
00258                             case Tags.ColourClose:
00259                                 if (brushes.Count > 1)
00260                                 {
00261                                     if (!string.IsNullOrEmpty(txt))
00262                                     {
00263                                         yield return new FormattedText(txt, GetFont(boldDepth % 2 ==
      1, italicsDepth % 2 == 1, underlineDepth % 2 == 1, normalFont, boldFont, italicFont, boldItalicFont),
      superscriptDepth > subscriptDepth ?  Script.Superscript :  subscriptDepth > superscriptDepth ?
      Script.Subscript :  Script.Normal, brushes.Peek(), spacing);
00264                                     }
00265                                     currentRun.Clear();
00266                                     brushes.Pop();
00267                                 }
00268                                 break;
00269                         }
00270                     }
00271                 }
00272             }
00273
00274             if (currentRun.Length > 0)
00275             {
00276                 yield return new FormattedText(currentRun.ToString(), GetFont(boldDepth % 2 == 1,
```

```
italicsDepth % 2 == 1, underlineDepth % 2 == 1, normalFont, boldFont, italicFont, boldItalicFont),
superscriptDepth > subscriptDepth ?  Script.Superscript :  subscriptDepth > superscriptDepth ?
Script.Subscript :  Script.Normal, brushes.Peek(), spacing);
00277             }
00278         }
00279
00280 /// <summary>
00281 /// Parse the formatting information contained in a text string into a collection of <see
cref="FormattedText"/> objects, using fonts from a standard font family.
00282 /// </summary>
00283 /// <param name="text">The string containing formatting information.  Format information is specified
using HTML-like tags:
00284 /// <list type="bullet">
00285 /// <item><c>&lt;b&gt;&lt;/b&gt;</c> or <c>&lt;strong&gt;&lt;/strong&gt;</c> are used for bold
text;</item>
00286 /// <item><c>&lt;i&gt;&lt;/i&gt;</c> or <c>&lt;em&gt;&lt;/em&gt;</c> are used for text in
italics;</item>
00287 /// <item><c>&lt;u&gt;&lt;/u&gt;</c></item> are used for underlined text;
00288 /// <item><c>&lt;sup&gt;&lt;/sup&gt;</c> and <c>&lt;sub&gt;&lt;/sub&gt;</c> are used, respectively,
for superscript and subscript text;</item>
00289 /// <item><c>&lt;#COLOUR&gt;&lt;/#&gt;</c> is used to specify the colour of the text, where
<c>COLOUR</c> is a CSS colour string (e.g.  <c>&lt;#red&gt;</c>, <c>&lt;#0080FF&gt;</c>, or
<c>&lt;#rgba(128, 80, 52, 0.5)&gt;</c>).</item>
00290 /// </list></param>
00291 /// <param name="fontFamily">The font family from which the fonts will be created.  If this is a
regular font family, the bold, italic and bold-italic versions of the font will be used for the
formatted text.  Otherwise, the relevant font styles will be toggled (e.g.  if the supplied font
family is bold, then regular text in the formatted string will be displayed as bold, while bold text
in the formatted string will be displayed as regular text).</param>
00292 /// <param name="fontSize">The size of the fonts to use.</param>
00293 /// <param name="defaultUnderline">Determines whether text should be underlined by default.  This is
toggled by <c>&lt;u&gt;&lt;/u&gt;</c> tags.</param>
00294 /// <param name="defaultBrush">The default <see cref="Brush"/> that will be used for text runs that do
not specify a colour.  If this is <see langword="null"/>, the default <see cref="Brush"/> will be the
one specified in the painting call.</param>
00295 /// <param name="spacing">The text spacing parameters used to draw the text.</param>
00296 /// <returns>A lazy collection of <see cref="FormattedText"/> objects.  Note that every enumeration of
this collection causes the text to be parsed again; if you need to enumerate this collection more than
once, you should probably convert it e.g.  to a <see cref="List{T}"/>.</returns>
00297         public static IEnumerable<FormattedText> Format(string text, FontFamily.StandardFontFamilies
fontFamily, double fontSize, bool defaultUnderline = false, Brush defaultBrush = null, TextSpacing?
spacing = null)
00298         {
00299             Font normalFont = new Font(FontFamily.ResolveFontFamily(fontFamily), fontSize,
defaultUnderline);
00300
00301             Font boldFont = normalFont;
00302             Font italicFont = normalFont;
00303             Font boldItalicFont = normalFont;
00304
00305             switch (fontFamily)
00306             {
00307                 case FontFamily.StandardFontFamilies.Courier:
00308                     boldFont = new
Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierBold), fontSize,
defaultUnderline);
00309                     italicFont = new
Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierOblique), fontSize,
defaultUnderline);
00310                     boldItalicFont = new
Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierBoldOblique), fontSize,
defaultUnderline);
00311                     break;
00312                 case FontFamily.StandardFontFamilies.CourierBold:
00313                     boldFont = new
Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Courier), fontSize,
defaultUnderline);
00314                     italicFont = new
Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierBoldOblique), fontSize,
defaultUnderline);
00315                     boldItalicFont = new
Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierOblique), fontSize,
defaultUnderline);
00316                     break;
00317                 case FontFamily.StandardFontFamilies.CourierOblique:
00318                     boldFont = new
Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierBoldOblique), fontSize,
defaultUnderline);
00319                     italicFont = new
Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Courier), fontSize,
defaultUnderline);
00320                     boldItalicFont = new
Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierBold), fontSize,
defaultUnderline);
00321                     break;
00322                 case FontFamily.StandardFontFamilies.CourierBoldOblique:
00323                     boldFont = new
```

```
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierOblique), fontSize,
      defaultUnderline);
00324                     italicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.CourierBold), fontSize,
      defaultUnderline);
00325                     boldItalicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Courier), fontSize,
      defaultUnderline);
00326                     break;
00327
00328                 case FontFamily.StandardFontFamilies.Helvetica:
00329                     boldFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), fontSize,
      defaultUnderline);
00330                     italicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaOblique), fontSize,
      defaultUnderline);
00331                     boldItalicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBoldOblique), fontSize,
      defaultUnderline);
00332                     break;
00333                 case FontFamily.StandardFontFamilies.HelveticaBold:
00334                     boldFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica), fontSize,
      defaultUnderline);
00335                     italicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBoldOblique), fontSize,
      defaultUnderline);
00336                     boldItalicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaOblique), fontSize,
      defaultUnderline);
00337                     break;
00338                 case FontFamily.StandardFontFamilies.HelveticaOblique:
00339                     boldFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBoldOblique), fontSize,
      defaultUnderline);
00340                     italicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica), fontSize,
      defaultUnderline);
00341                     boldItalicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), fontSize,
      defaultUnderline);
00342                     break;
00343                 case FontFamily.StandardFontFamilies.HelveticaBoldOblique:
00344                     boldFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaOblique), fontSize,
      defaultUnderline);
00345                     italicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.HelveticaBold), fontSize,
      defaultUnderline);
00346                     boldItalicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.Helvetica), fontSize,
      defaultUnderline);
00347                     break;
00348
00349                 case FontFamily.StandardFontFamilies.TimesRoman:
00350                     boldFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesBold), fontSize,
      defaultUnderline);
00351                     italicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesItalic), fontSize,
      defaultUnderline);
00352                     boldItalicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesBoldItalic), fontSize,
      defaultUnderline);
00353                     break;
00354                 case FontFamily.StandardFontFamilies.TimesBold:
00355                     boldFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesRoman), fontSize,
      defaultUnderline);
00356                     italicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesBoldItalic), fontSize,
      defaultUnderline);
00357                     boldItalicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesItalic), fontSize,
      defaultUnderline);
00358                     break;
00359                 case FontFamily.StandardFontFamilies.TimesItalic:
00360                     boldFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesBoldItalic), fontSize,
      defaultUnderline);
00361                     italicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesRoman), fontSize,
      defaultUnderline);
00362                     boldItalicFont = new
      Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesBold), fontSize,
      defaultUnderline);
```

```
00363                   break;
00364               case FontFamily.StandardFontFamilies.TimesBoldItalic:
00365                   boldFont = new
    Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesItalic), fontSize,
    defaultUnderline);
00366                   italicFont = new
    Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesBold), fontSize,
    defaultUnderline);
00367                   boldItalicFont = new
    Font(FontFamily.ResolveFontFamily(FontFamily.StandardFontFamilies.TimesRoman), fontSize,
    defaultUnderline);
00368                   break;
00369           }
00370
00371           return Format(text, normalFont, boldFont, italicFont, boldItalicFont, defaultBrush,
    spacing);
00372       }
00373
00374       private static Font GetFont(bool bold, bool italic, bool underlined, Font normalFont, Font
    boldFont, Font italicFont, Font boldItalicFont)
00375       {
00376           if (!bold && !italic)
00377           {
00378               if (underlined)
00379               {
00380                   return new Font(normalFont.FontFamily, normalFont.FontSize, normalFont.Underline
    == null);
00381               }
00382               else
00383               {
00384                   return normalFont;
00385               }
00386           }
00387           else if (bold && !italic)
00388           {
00389               if (underlined)
00390               {
00391                   return new Font(boldFont.FontFamily, boldFont.FontSize, boldFont.Underline ==
    null);
00392               }
00393               else
00394               {
00395                   return boldFont;
00396               }
00397           }
00398           else if (!bold && italic)
00399           {
00400               if (underlined)
00401               {
00402                   return new Font(italicFont.FontFamily, italicFont.FontSize, italicFont.Underline
    == null);
00403               }
00404               else
00405               {
00406                   return italicFont;
00407               }
00408           }
00409           else
00410           {
00411               if (underlined)
00412               {
00413                   return new Font(boldItalicFont.FontFamily, boldItalicFont.FontSize,
    boldItalicFont.Underline == null);
00414               }
00415               else
00416               {
00417                   return boldItalicFont;
00418               }
00419           }
00420       }
00421
00422       private enum Tags
00423       {
00424           BoldOpen,
00425           BoldClose,
00426           ItalicsOpen,
00427           ItalicsClose,
00428           UnderlineOpen,
00429           UnderlineClose,
00430           SupOpen,
00431           SupClose,
00432           SubOpen,
00433           SubClose,
00434           ColourOpen,
00435           ColourClose,
00436           None
00437       }
```

```
00438
00439         private static Tags GetTag(string text, int start, out int tagEnd, out Brush tagBrush)
00440         {
00441             StringBuilder tag = new StringBuilder();
00442             bool closed = false;
00443
00444             int i = start;
00445
00446             for (; i < text.Length; i++)
00447             {
00448                 tag.Append(text[i]);
00449
00450                 if (text[i] == '>')
00451                 {
00452                     closed = true;
00453                     break;
00454                 }
00455             }
00456
00457             tagEnd = i;
00458             tagBrush = null;
00459
00460             if (!closed)
00461             {
00462                 return Tags.None;
00463             }
00464             else
00465             {
00466                 string tagString = tag.Replace(" ", "").ToString().ToLowerInvariant();
00467
00468                 if (tagString == "<b>" || tagString == "<strong>")
00469                 {
00470                     return Tags.BoldOpen;
00471                 }
00472                 else if (tagString == "</b>" || tagString == "</strong>")
00473                 {
00474                     return Tags.BoldClose;
00475                 }
00476                 else if (tagString == "<i>" || tagString == "<em>")
00477                 {
00478                     return Tags.ItalicsOpen;
00479                 }
00480                 else if (tagString == "</i>" || tagString == "</em>")
00481                 {
00482                     return Tags.ItalicsClose;
00483                 }
00484                 else if (tagString == "<u>")
00485                 {
00486                     return Tags.UnderlineOpen;
00487                 }
00488                 else if (tagString == "</u>")
00489                 {
00490                     return Tags.UnderlineClose;
00491                 }
00492                 else if (tagString == "<sup>")
00493                 {
00494                     return Tags.SupOpen;
00495                 }
00496                 else if (tagString == "</sup>")
00497                 {
00498                     return Tags.SupClose;
00499                 }
00500                 else if (tagString == "<sub>")
00501                 {
00502                     return Tags.SubOpen;
00503                 }
00504                 else if (tagString == "</sub>")
00505                 {
00506                     return Tags.SubClose;
00507                 }
00508                 else if (tagString.StartsWith("<#"))
00509                 {
00510                     string colour = tagString.Substring(1, tagString.Length - 2);
00511
00512                     Colour?  col = null;
00513
00514                     try
00515                     {
00516                         col = Colour.FromCSSString(colour);
00517                     }
00518                     catch { }
00519
00520                     if (col == null)
00521                     {
00522                         colour = tagString.Substring(2, tagString.Length - 3);
00523
00524                         col = Colour.FromCSSString(colour);
```

```
00525                     }
00526
00527                     if (col != null)
00528                     {
00529                         tagBrush = col.Value;
00530                         return Tags.ColourOpen;
00531                     }
00532                     else
00533                     {
00534                         return Tags.None;
00535                     }
00536                 }
00537                 else if (tagString == "</#>")
00538                 {
00539                     return Tags.ColourClose;
00540                 }
00541                 else
00542                 {
00543                     return Tags.None;
00544                 }
00545             }
00546         }
00547     }
00548
00549 /// <summary>
00550 /// Contains extension methods for collections of <see cref="FormattedText"/> objects.
00551 /// </summary>
00552     public static class FormattedTextExtensions
00553     {
00554         internal static Font.DetailedFontMetrics Measure(this IEnumerable<FormattedText> text,
      List<FormattedText> items, List<Font.DetailedFontMetrics> allMetrics)
00555         {
00556             double width = 0;
00557             double advanceWidth = 0;
00558
00559             double lsb = 0;
00560             double rsb = 0;
00561
00562             double top = 0;
00563             double bottom = 0;
00564
00565             bool isFirst = true;
00566
00567             FormattedText[] fText = text.ToArray();
00568
00569             for (int i = 0; i < fText.Length; i++)
00570             {
00571                 FormattedText txt = fText[i];
00572
00573                 if (text != null && !string.IsNullOrEmpty(txt.Text))
00574                 {
00575                     items?.Add(txt);
00576
00577                     if (txt.Script == Script.Normal)
00578                     {
00579                         Font.DetailedFontMetrics metrics;
00580
00581                         if (txt.Spacing == TextSpacing.Default)
00582                         {
00583                             metrics = txt.Font.MeasureTextAdvanced(txt.Text);
00584                         }
00585                         else
00586                         {
00587                             metrics = Graphics.MeasureTextWithSpacing(0, 0, txt.Text, txt.Font, i ==
      fText.Length - 1, txt.Spacing);
00588                         }
00589
00590                         allMetrics?.Add(metrics);
00591
00592                         top = Math.Max(top, metrics.Top);
00593                         bottom = Math.Min(bottom, metrics.Bottom);
00594                         rsb = metrics.RightSideBearing;
00595
00596                         advanceWidth += metrics.AdvanceWidth;
00597
00598                         if (!isFirst)
00599                         {
00600                             width += metrics.Width + metrics.RightSideBearing +
      metrics.LeftSideBearing;
00601                         }
00602                         else
00603                         {
00604                             width += metrics.Width + metrics.RightSideBearing;
00605                             lsb = metrics.LeftSideBearing;
00606                         }
00607                     }
00608                     else
```

```
00609                     {
00610                         Font newFont = new Font(txt.Font.FontFamily, txt.Font.FontSize * 0.7);
00611 
00612                         Font.DetailedFontMetrics metrics;
00613 
00614                         if (txt.Spacing == TextSpacing.Default)
00615                         {
00616                             metrics = newFont.MeasureTextAdvanced(txt.Text);
00617                         }
00618                         else
00619                         {
00620                             metrics = Graphics.MeasureTextWithSpacing(0, 0, txt.Text, newFont, i ==
      fText.Length - 1, txt.Spacing);
00621                         }
00622 
00623                         allMetrics?.Add(metrics);
00624 
00625                         advanceWidth += metrics.AdvanceWidth;
00626 
00627                         if (txt.Script == Script.Subscript)
00628                         {
00629                             top = Math.Max(top, metrics.Top - txt.Font.FontSize * 0.14);
00630                             bottom = Math.Min(bottom, metrics.Bottom - txt.Font.FontSize * 0.14);
00631                         }
00632                         else if (txt.Script == Script.Superscript)
00633                         {
00634                             top = Math.Max(top, metrics.Top + txt.Font.FontSize * 0.33);
00635                             bottom = Math.Min(bottom, metrics.Bottom + txt.Font.FontSize * 0.33);
00636                         }
00637 
00638                         rsb = metrics.RightSideBearing;
00639 
00640                         if (!isFirst)
00641                         {
00642                             width += metrics.Width + metrics.RightSideBearing +
      metrics.LeftSideBearing;
00643                         }
00644                         else
00645                         {
00646                             width += metrics.Width + metrics.RightSideBearing;
00647                             lsb = metrics.LeftSideBearing;
00648                         }
00649                     }
00650 
00651                     if (isFirst)
00652                     {
00653                         isFirst = false;
00654                     }
00655                 }
00656             }
00657 
00658             width -= rsb;
00659 
00660             return new Font.DetailedFontMetrics(width, top - bottom, lsb, rsb, top, bottom,
      advanceWidth);
00661         }
00662 
00663 /// <summary>
00664 /// Measures a collection of <see cref="FormattedText"/> objects.
00665 /// </summary>
00666 /// <param name="text">The collection of <see cref="FormattedText"/> objects to be measured.</param>
00667 /// <returns>A <see cref="Font.DetailedFontMetrics"/> containing detailed measurements for the text
      obtained by composing the elements in the <see cref="FormattedText"/> collection.</returns>
00668         public static Font.DetailedFontMetrics Measure(this IEnumerable<FormattedText> text)
00669         {
00670             return text.Measure(null, null);
00671         }
00672 
00673 /// <summary>
00674 /// Extracts the text from a collection of <see cref="FormattedText"/> objects.
00675 /// </summary>
00676 /// <param name="text">The collection of <see cref="FormattedText"/> objects whose text should be
      extracted.</param>
00677 /// <returns>A text rappresentation of the collection of <see cref="FormattedText"/>
      objects.</returns>
00678         public static string GetText(this IEnumerable<FormattedText> text)
00679         {
00680             StringBuilder sb = new StringBuilder();
00681 
00682             foreach (FormattedText txt in text)
00683             {
00684                 sb.Append(txt.ToString());
00685             }
00686 
00687             return sb.ToString();
00688         }
00689     }
```

```
00690
00691 }
```

8.82 Gradients.cs

```
00001 using System;
00002 using System.Collections.Generic;
00003 using System.Linq;
00004 using System.Text;
00005 using VectSharp;
00006 using static System.Net.WebRequestMethods;
00007
00008 namespace VectSharp
00009 {
00010 /// <summary>
00011 /// Standard gradients.
00012 /// </summary>
00013     public static class Gradients
00014     {
00015 /// <summary>
00016 /// Gradient from transparent black (0) to opaque black (1).
00017 /// </summary>
00018         public static readonly GradientStops TransparentToBlack = new GradientStops(new
      GradientStop(Colour.FromRgba(0, 0, 0, 0), 0), new GradientStop(Colour.FromRgb(0, 0, 0), 1));
00019
00020 /// <summary>
00021 /// Gradient from white (0) to black (1).
00022 /// </summary>
00023         public static readonly GradientStops WhiteToBlack = new GradientStops(new
      GradientStop(Colour.FromRgb(255, 255, 255), 0), new GradientStop(Colour.FromRgb(0, 0, 0), 1));
00024
00025 /// <summary>
00026 /// Gradient from red (0) to green (1).
00027 /// </summary>
00028         public static readonly GradientStops RedToGreen = new GradientStops(new
      GradientStop(Colour.FromRgb(237, 28, 36), 0), new GradientStop(Colour.FromRgb(34, 177, 76), 1));
00029
00030 /// <summary>
00031 /// Rainbow gradient (red, orange, yellow, green, blue, indigo, violet).
00032 /// </summary>
00033         public static readonly GradientStops Rainbow = new GradientStops(new
      GradientStop(Colour.FromRgb(237, 28, 36), 0), new GradientStop(Colour.FromRgb(255, 127, 39), 1.0 / 6),
      new GradientStop(Colour.FromRgb(255, 242, 0), 1.0 / 3), new GradientStop(Colour.FromRgb(34, 177, 76),
      0.5), new GradientStop(Colour.FromRgb(0, 162, 232), 2.0 / 3), new GradientStop(Colour.FromRgb(63, 72,
      204), 5.0 / 6), new GradientStop(Colour.FromRgb(163, 73, 164), 1));
00034
00035 /// <summary>
00036 /// Gradient from red (0) to yellow (0.5) to green (1).
00037 /// </summary>
00038         public static readonly GradientStops RedYellowGreen = new GradientStops(new
      GradientStop(Colour.FromRgb(237, 28, 36), 0), new GradientStop(Colour.FromRgb(255, 242, 0), 0.5), new
      GradientStop(Colour.FromRgb(34, 177, 76), 1));
00039
00040 /// <summary>
00041 /// Rainbow gradient with Okabe-Ito colour-blind safe colours (<seealso
      href="https://jfly.uni-koeln.de/color/"/>).
00042 /// </summary>
00043         public static readonly GradientStops OkabeItoRainbow = new GradientStops(new
      GradientStop(Colour.FromRgb(204, 121, 167), 0), new GradientStop(Colour.FromRgb(213, 94, 0), 1.0 / 6),
      new GradientStop(Colour.FromRgb(230, 159, 0), 1.0 / 3), new GradientStop(Colour.FromRgb(240, 228, 66),
      0.5), new GradientStop(Colour.FromRgb(0, 158, 115), 2.0 / 3), new GradientStop(Colour.FromRgb(0, 114,
      178), 5.0 / 6), new GradientStop(Colour.FromRgb(86, 180, 233), 1));
00044
00045 /// <summary>
00046 /// Rainbow gradient with Okabe-Ito colour-blind safe colours (<seealso
      href="https://jfly.uni-koeln.de/color/"/>) in discrete steps.
00047 /// </summary>
00048         public static readonly GradientStops OkabeItoRainbowDiscrete = new GradientStops(new
      GradientStop(Colour.FromRgb(204, 121, 167), 0), new GradientStop(Colour.FromRgb(204, 121, 167), 0.2),
      new GradientStop(Colour.FromRgb(230, 159, 0), 0.201), new GradientStop(Colour.FromRgb(230, 159, 0),
      0.4), new GradientStop(Colour.FromRgb(0, 158, 115), 0.401), new GradientStop(Colour.FromRgb(0, 158,
      115), 0.6), new GradientStop(Colour.FromRgb(0, 114, 178), 0.601), new GradientStop(Colour.FromRgb(0,
      114, 178), 0.8), new GradientStop(Colour.FromRgb(86, 180, 233), 0.801), new
      GradientStop(Colour.FromRgb(86, 180, 233), 1));
00049
00050 /// <summary>
00051 /// Rainbow gradient with Paul Tol's Muted palette (<seealso
      href="https://personal.sron.nl/~pault/"/>).
00052 /// </summary>
00053         public static readonly GradientStops MutedRainbow = new GradientStops(new
      GradientStop(Colour.FromRgb(221, 204, 119), 0), new GradientStop(Colour.FromRgb(153, 153, 51), 0.125),
      new GradientStop(Colour.FromRgb(17, 119, 51), 0.25), new GradientStop(Colour.FromRgb(68, 170, 153),
      0.375), new GradientStop(Colour.FromRgb(136, 204, 238), 0.5), new GradientStop(Colour.FromRgb(51, 34,
```

```
      136), 0.625), new GradientStop(Colour.FromRgb(170, 68, 153), 0.75), new
      GradientStop(Colour.FromRgb(136, 34, 85), 0.875), new GradientStop(Colour.FromRgb(204, 102, 119), 1));
00054
00055 /// <summary>
00056 /// Rainbow gradient with Paul Tol's Muted palette (<seealso
      href="https://personal.sron.nl/~pault/"/>) in discrete steps.
00057 /// </summary>
00058         public static readonly GradientStops MutedRainbowDiscrete = new GradientStops(new
      GradientStop(Colour.FromRgb(187, 187, 187), 0.101), new GradientStop(Colour.FromRgb(187, 187, 187),
      0.1), new GradientStop(Colour.FromRgb(221, 204, 119), 0.101), new GradientStop(Colour.FromRgb(221,
      204, 119), 0.2), new GradientStop(Colour.FromRgb(153, 153, 51), 0.201), new
      GradientStop(Colour.FromRgb(153, 153, 51), 0.3), new GradientStop(Colour.FromRgb(17, 119, 51), 0.301),
      new GradientStop(Colour.FromRgb(17, 119, 51), 0.4), new GradientStop(Colour.FromRgb(68, 170, 153),
      0.401), new GradientStop(Colour.FromRgb(68, 170, 153), 0.5), new GradientStop(Colour.FromRgb(136, 204,
      238), 0.501), new GradientStop(Colour.FromRgb(136, 204, 238), 0.6), new
      GradientStop(Colour.FromRgb(51, 34, 136), 0.601), new GradientStop(Colour.FromRgb(51, 34, 136), 0.7),
      new GradientStop(Colour.FromRgb(170, 68, 153), 0.701), new GradientStop(Colour.FromRgb(170, 68, 153),
      0.8), new GradientStop(Colour.FromRgb(136, 34, 85), 0.801), new GradientStop(Colour.FromRgb(136, 34,
      85), 0.9), new GradientStop(Colour.FromRgb(204, 102, 119), 0.901), new
      GradientStop(Colour.FromRgb(204, 102, 119), 1));
00059
00060         private static double[,] MagmaRGB = new double[256, 3] { { 0.00146159096, 0.000466127766,
      0.01386552 }, { 0.00225764007, 0.00129495431, 0.0183311461 }, { 0.00327943222, 0.00230452991,
      0.0237083291 }, { 0.00451230222, 0.00349037666, 0.0299647059 }, { 0.00594976987, 0.00484285,
      0.0371296695 }, { 0.0075879855, 0.00635613622, 0.0449730774 }, { 0.0094260439, 0.00802185006,
      0.0528443561 }, { 0.0114654337, 0.00982831486, 0.060749638 }, { 0.0137075706, 0.0117705913,
      0.0686665843 }, { 0.0161557566, 0.0138404966, 0.076602666 }, { 0.018815367, 0.0160262753, 0.0845844897
      }, { 0.021691934, 0.0183201254, 0.092610105 }, { 0.0247917814, 0.0207147875, 0.100675555 }, {
      0.0281228154, 0.0232009284, 0.108786954 }, { 0.0316955304, 0.0257651161, 0.116964722 }, {
      0.0355204468, 0.028397457, 0.125209396 }, { 0.0396084872, 0.0310895652, 0.133515085 }, { 0.043829535,
      0.0338299885, 0.141886249 }, { 0.0480616391, 0.0366066101, 0.150326989 }, { 0.0523204388, 0.039406602,
      0.158841025 }, { 0.0566148978, 0.0421598925, 0.167445592 }, { 0.060949393, 0.0447944924, 0.176128834
      }, { 0.0653301801, 0.0473177796, 0.184891506 }, { 0.0697637296, 0.0497264666, 0.193735088 }, {
      0.0742565152, 0.0520167766, 0.202660374 }, { 0.0788150034, 0.0541844801, 0.211667355 }, {
      0.0834456313, 0.0562249365, 0.220755099 }, { 0.088154773, 0.0581331465, 0.229921611 }, { 0.0929486914,
      0.0599038167, 0.239163669 }, { 0.097833477, 0.0615314414, 0.248476662 }, { 0.102814972, 0.0630104053,
      0.2578544 }, { 0.107898679, 0.0643351102, 0.267288933 }, { 0.113094451, 0.0654920358, 0.276783978 }, {
      0.118405035, 0.0664791593, 0.286320656 }, { 0.123832651, 0.0672946449, 0.295879431 }, { 0.129380192,
      0.0679349264, 0.305442931 }, { 0.135053322, 0.0683912798, 0.31499989 }, { 0.140857952, 0.068654071,
      0.32453764 }, { 0.146785234, 0.0687382323, 0.334011109 }, { 0.152839217, 0.0686368599, 0.34340445 }, {
      0.159017511, 0.0683540225, 0.352688028 }, { 0.165308131, 0.0679108689, 0.361816426 }, { 0.171713033,
      0.067305326, 0.370770827 }, { 0.17821173, 0.0665758073, 0.379497161 }, { 0.184800877, 0.0657324381,
      0.387972507 }, { 0.191459745, 0.0648183312, 0.396151969 }, { 0.198176877, 0.0638624166, 0.404008953 },
      { 0.204934882, 0.0629066192, 0.411514273 }, { 0.211718061, 0.0619917876, 0.418646741 }, { 0.21851159,
      0.0611584918, 0.425391816 }, { 0.225302032, 0.0604451843, 0.431741767 }, { 0.232076515, 0.0598886855,
      0.437694665 }, { 0.238825991, 0.0595170384, 0.443255999 }, { 0.245543175, 0.0593524384, 0.448435938 },
      { 0.252220252, 0.0594147119, 0.453247729 }, { 0.258857304, 0.0597055998, 0.457709924 }, { 0.265446744,
      0.0602368754, 0.461840297 }, { 0.271994089, 0.0609935552, 0.465660375 }, { 0.2784933, 0.0619778136,
      0.469190328 }, { 0.284951097, 0.0631676261, 0.472450879 }, { 0.291365817, 0.0645534486, 0.475462193 },
      { 0.297740413, 0.0661170432, 0.478243482 }, { 0.304080941, 0.0678353452, 0.480811572 }, { 0.310382027,
      0.0697024767, 0.48318634 }, { 0.316654235, 0.0716895272, 0.485380429 }, { 0.322899126, 0.0737819504,
      0.487408399 }, { 0.329114038, 0.0759715081, 0.489286796 }, { 0.335307503, 0.0782361045, 0.491024144 },
      { 0.341481725, 0.0805635079, 0.492631321 }, { 0.347635742, 0.0829463512, 0.494120923 }, { 0.353773161,
      0.0853726329, 0.495501096 }, { 0.359897941, 0.0878311772, 0.496778331 }, { 0.366011928, 0.0903143031,
      0.497959963 }, { 0.372116205, 0.0928159917, 0.499053326 }, { 0.378210547, 0.0953322947, 0.500066568 },
      { 0.384299445, 0.0978549106, 0.501001964 }, { 0.390384361, 0.100379466, 0.501864236 }, { 0.39646667,
      0.102902194, 0.50265759 }, { 0.402547663, 0.105419865, 0.503385761 }, { 0.408628505, 0.107929771,
      0.504052118 }, { 0.414708664, 0.110431177, 0.504661843 }, { 0.420791157, 0.11292021, 0.505214935 }, {
      0.426876965, 0.115395258, 0.505713602 }, { 0.432967001, 0.117854987, 0.506159754 }, { 0.439062114,
      0.120298314, 0.506555026 }, { 0.445163096, 0.122724371, 0.506900806 }, { 0.451270678, 0.125132484,
      0.507198258 }, { 0.457385535, 0.127522145, 0.507448336 }, { 0.463508291, 0.129892998, 0.507651812 }, {
      0.469639514, 0.132244819, 0.507809282 }, { 0.475779723, 0.1345775, 0.507921193 }, { 0.481928997,
      0.13689139, 0.507988509 }, { 0.488088169, 0.139186217, 0.508010737 }, { 0.494257673, 0.141462106,
      0.507987836 }, { 0.500437834, 0.143719323, 0.507919772 }, { 0.506628929, 0.145958202, 0.50780642 }, {
      0.512831195, 0.148179144, 0.50764757 }, { 0.519044825, 0.150382611, 0.507442938 }, { 0.525269968,
      0.152569121, 0.507192172 }, { 0.531506735, 0.154739247, 0.50689486 }, { 0.537755194, 0.156893613,
      0.506550538 }, { 0.544015371, 0.159032895, 0.506158696 }, { 0.550287252, 0.161157816, 0.505718782 }, {
      0.556570783, 0.163269149, 0.50523021 }, { 0.562865867, 0.165367714, 0.504692365 }, { 0.569172368,
      0.167454379, 0.504104606 }, { 0.575490107, 0.169530062, 0.503466273 }, { 0.581818864, 0.171595728,
      0.50277669 }, { 0.588158375, 0.173652392, 0.502035167 }, { 0.594508337, 0.175701122, 0.501241011 }, {
      0.600868399, 0.177743036, 0.500393522 }, { 0.607238169, 0.179779309, 0.499491999 }, { 0.613617209,
      0.18181117, 0.498535746 }, { 0.620005032, 0.183839907, 0.497524075 }, { 0.626401108, 0.185866869,
      0.496456304 }, { 0.632804854, 0.187893468, 0.495331769 }, { 0.639215638, 0.189921182, 0.494149821 }, {
      0.645632778, 0.191951556, 0.492909832 }, { 0.652055535, 0.19398621, 0.491611196 }, { 0.658483116,
      0.196026835, 0.490253338 }, { 0.664914668, 0.198075202, 0.488835712 }, { 0.671349279, 0.200133166,
      0.487357807 }, { 0.677785975, 0.202202663, 0.485819154 }, { 0.684223712, 0.204285721, 0.484219325 }, {
      0.69066138, 0.206384461, 0.482557941 }, { 0.697097796, 0.2085011, 0.480834678 }, { 0.7035317,
      0.210637956, 0.47904927 }, { 0.709961888, 0.212797337, 0.477201121 }, { 0.716387038, 0.214981693,
      0.47528978 }, { 0.722805451, 0.217193831, 0.473315708 }, { 0.729215521, 0.219436516, 0.471278924 }, {
      0.735615545, 0.221712634, 0.469179541 }, { 0.742003713, 0.224025196, 0.467017774 }, { 0.748378107,
      0.226377345, 0.464793954 }, { 0.754736692, 0.228772352, 0.462508534 }, { 0.761077312, 0.231213625,
      0.460162106 }, { 0.767397681, 0.233704708, 0.457755411 }, { 0.77369538, 0.236249283, 0.455289354 }, {
      0.779967847, 0.23885117, 0.452765022 }, { 0.786212372, 0.241514325, 0.450183695 }, { 0.792426972,
      0.24424225, 0.447543155 }, { 0.79860776, 0.247039798, 0.444848441 }, { 0.804751511, 0.24991135,
      0.442101615 }, { 0.810854841, 0.252861399, 0.439304963 }, { 0.816914186, 0.25589455, 0.436461074 }, {
      0.822925797, 0.259015505, 0.433572874 }, { 0.82888574, 0.262229049, 0.430643647 }, { 0.834790818,
      0.265539703, 0.427671352 }, { 0.84063568, 0.268952874, 0.42466562 }, { 0.846415804, 0.272473491,
```

```
0.421631064 }, { 0.85212649, 0.276106469, 0.418572767 }, { 0.85776287, 0.279856666, 0.415496319 }, {
0.863320397, 0.283729003, 0.412402889 }, { 0.868793368, 0.287728205, 0.409303002 }, { 0.874176342,
0.291858679, 0.406205397 }, { 0.879463944, 0.296124596, 0.403118034 }, { 0.884650824, 0.30053009,
0.40004706 }, { 0.889731418, 0.305078817, 0.397001559 }, { 0.894700194, 0.309773445, 0.393994634 }, {
0.899551884, 0.314616425, 0.391036674 }, { 0.904281297, 0.319609981, 0.388136889 }, { 0.908883524,
0.324755126, 0.385308008 }, { 0.913354091, 0.330051947, 0.382563414 }, { 0.917688852, 0.335500068,
0.379915138 }, { 0.921884187, 0.341098112, 0.377375977 }, { 0.925937102, 0.346843685, 0.374959077 }, {
0.92984509, 0.352733817, 0.372676513 }, { 0.933606454, 0.358764377, 0.370540883 }, { 0.937220874,
0.364929312, 0.368566525 }, { 0.940687443, 0.371224168, 0.366761699 }, { 0.944006448, 0.377642889,
0.365136328 }, { 0.947179528, 0.384177874, 0.36370113 }, { 0.95021015, 0.390819546, 0.362467694 }, {
0.953099077, 0.397562894, 0.361438431 }, { 0.955849237, 0.404400213, 0.360619076 }, { 0.958464079,
0.411323666, 0.360014232 }, { 0.960949221, 0.418323245, 0.359629789 }, { 0.963310281, 0.425389724,
0.35946902 }, { 0.965549351, 0.432518707, 0.359529151 }, { 0.967671128, 0.439702976, 0.359810172 }, {
0.969680441, 0.446935635, 0.36031112 }, { 0.971582181, 0.45421017, 0.361030156 }, { 0.973381238,
0.461520484, 0.361964652 }, { 0.975082439, 0.468860936, 0.363111292 }, { 0.976690494, 0.47622635,
0.364466162 }, { 0.978209957, 0.483612031, 0.366024854 }, { 0.979645181, 0.491013764, 0.367782559 }, {
0.981000291, 0.4984278, 0.369734157 }, { 0.982279159, 0.505850848, 0.371874301 }, { 0.983485387,
0.513280054, 0.374197501 }, { 0.984622298, 0.520712972, 0.376698186 }, { 0.985692925, 0.528147545,
0.379370774 }, { 0.986700017, 0.53558207, 0.382209724 }, { 0.987646038, 0.543015173, 0.385209578 }, {
0.988533173, 0.550445778, 0.388365009 }, { 0.989363341, 0.557873075, 0.391670846 }, { 0.990138201,
0.565296495, 0.395122099 }, { 0.990871208, 0.572706259, 0.398713971 }, { 0.991558165, 0.580106828,
0.402441058 }, { 0.992195728, 0.587501706, 0.406298792 }, { 0.992784669, 0.594891088, 0.410282976 }, {
0.993325561, 0.602275297, 0.414389658 }, { 0.993834412, 0.60964354, 0.418613221 }, { 0.994308514,
0.616998953, 0.422949672 }, { 0.994737698, 0.624349657, 0.427396771 }, { 0.995121854, 0.631696376,
0.431951492 }, { 0.995480469, 0.639026596, 0.436607159 }, { 0.995809924, 0.646343897, 0.441360951 }, {
0.996095703, 0.653658756, 0.446213021 }, { 0.996341406, 0.660969379, 0.451160201 }, { 0.996579803,
0.668255621, 0.456191814 }, { 0.996774784, 0.675541484, 0.461314158 }, { 0.996925427, 0.682827953,
0.466525689 }, { 0.997077185, 0.690087897, 0.471811461 }, { 0.997186253, 0.697348991, 0.477181727 }, {
0.997253982, 0.704610791, 0.482634651 }, { 0.99732518, 0.711847714, 0.488154375 }, { 0.997350983,
0.719089119, 0.493754665 }, { 0.997350583, 0.726324415, 0.499427972 }, { 0.997341259, 0.733544671,
0.505166839 }, { 0.997284689, 0.740771893, 0.510983331 }, { 0.997228367, 0.747980563, 0.516859378 }, {
0.99713848, 0.755189852, 0.522805996 }, { 0.997019342, 0.762397883, 0.528820775 }, { 0.996898254,
0.769590975, 0.534892341 }, { 0.996726862, 0.77679486, 0.541038571 }, { 0.996570645, 0.783976508,
0.547232992 }, { 0.996369065, 0.791167346, 0.553498939 }, { 0.996162309, 0.798347709, 0.559819643 }, {
0.995932448, 0.805527126, 0.566201824 }, { 0.995680107, 0.812705773, 0.572644795 }, { 0.995423973,
0.819875302, 0.57914013 }, { 0.995131288, 0.827051773, 0.585701463 }, { 0.994851089, 0.834212826,
0.592307093 }, { 0.994523666, 0.841386618, 0.598982818 }, { 0.9942219, 0.848540474, 0.605695903 }, {
0.993865767, 0.855711038, 0.612481798 }, { 0.993545285, 0.862858846, 0.6192993 }, { 0.993169558,
0.870024467, 0.626189463 }, { 0.992830963, 0.877168404, 0.633109148 }, { 0.992439881, 0.884329694,
0.640099465 }, { 0.992089454, 0.891469549, 0.647116021 }, { 0.991687744, 0.89862705, 0.654201544 }, {
0.991331929, 0.905762748, 0.661308839 }, { 0.990929685, 0.91291501, 0.668481201 }, { 0.990569914,
0.920048699, 0.675674592 }, { 0.990174637, 0.927195612, 0.682925602 }, { 0.989814839, 0.93432854,
0.690198194 }, { 0.989433736, 0.941470354, 0.697518628 }, { 0.989077438, 0.948604077, 0.704862519 }, {
0.988717064, 0.95574152, 0.712242232 }, { 0.988367028, 0.962878026, 0.719648627 }, { 0.988032885,
0.970012413, 0.727076773 }, { 0.987690702, 0.977154231, 0.734536205 }, { 0.987386827, 0.984287561,
0.742001547 }, { 0.987052509, 0.991437853, 0.749504188 } };
00061         private static double[,] InfernoRGB = new double[256, 3] { { 0.00146159096, 0.000466127766,
0.01386552 }, { 0.00226726368, 0.00126992553, 0.018570352 }, { 0.00329899092, 0.00224934863,
0.0242390508 }, { 0.00454690615, 0.00339180156, 0.0309092475 }, { 0.00600552565, 0.00469194561,
0.038557898 }, { 0.00767578856, 0.00613611626, 0.0468360336 }, { 0.00956051094, 0.00771344131,
0.0551430756 }, { 0.0116634769, 0.00941675403, 0.063459808 }, { 0.0139950388, 0.0112247138,
0.071861689 }, { 0.0165605595, 0.0131362262, 0.0802817951 }, { 0.0193732295, 0.0151325789,
0.0887668094 }, { 0.0224468865, 0.0171991484, 0.0973274383 }, { 0.0257927373, 0.0193306298,
0.105929835 }, { 0.0294324251, 0.0215030771, 0.114621328 }, { 0.0333852235, 0.0237024271, 0.123397286
}, { 0.0376684211, 0.0259207864, 0.132232108 }, { 0.0422525554, 0.0281385015, 0.141140519 }, {
0.0469146287, 0.0303236129, 0.150163867 }, { 0.0516437624, 0.0324736172, 0.159254277 }, {
0.0564491009, 0.0345691867, 0.168413539 }, { 0.06133972, 0.0365900213, 0.177642172 }, { 0.066331262,
0.0385036268, 0.186961588 }, { 0.0714289181, 0.0402939095, 0.196353558 }, { 0.076636756, 0.0419053329,
0.205798788 }, { 0.0819620773, 0.0433278666, 0.215289113 }, { 0.0874113897, 0.0445561662, 0.224813479
}, { 0.0929901526, 0.0455829503, 0.234357604 }, { 0.0987024972, 0.0464018731, 0.2439037 }, {
0.104550936, 0.0470080541, 0.2534303 }, { 0.110536084, 0.0473986708, 0.262912235 }, { 0.116656423,
0.047573592, 0.272320803 }, { 0.122908126, 0.0475360183, 0.28162417 }, { 0.129284984, 0.0472930838,
0.290788012 }, { 0.13577845, 0.0468563678, 0.299776404 }, { 0.142377819, 0.0462422566, 0.30855291 }, {
0.149072957, 0.0454676444, 0.317085139 }, { 0.155849711, 0.0445588056, 0.325338414 }, { 0.162688939,
0.0435542881, 0.333276678 }, { 0.169575148, 0.0424893149, 0.340874188 }, { 0.176493202, 0.0414017089,
0.348110606 }, { 0.183428775, 0.0403288858, 0.354971391 }, { 0.190367453, 0.0393088888, 0.361446945 },
{ 0.197297425, 0.0384001825, 0.367534629 }, { 0.204209298, 0.0376322609, 0.373237557 }, { 0.211095463,
0.0370296488, 0.378563264 }, { 0.217948648, 0.0366146049, 0.383522415 }, { 0.224762908, 0.0364049901,
0.388128944 }, { 0.231538148, 0.0364052511, 0.39240015 }, { 0.238272961, 0.0366209949, 0.396353388 },
{ 0.244966911, 0.0370545017, 0.400006615 }, { 0.251620354, 0.0377052832, 0.403377897 }, { 0.258234265,
0.0385706153, 0.406485031 }, { 0.264809649, 0.0396468666, 0.409345373 }, { 0.271346664, 0.0409215821,
0.411976086 }, { 0.277849829, 0.0423528741, 0.414392106 }, { 0.284321318, 0.0439325787, 0.416607861 },
{ 0.290763373, 0.0456437598, 0.418636756 }, { 0.297178251, 0.0474700293, 0.420491164 }, { 0.303568182,
0.0493958927, 0.422182449 }, { 0.309935342, 0.0514069729, 0.423720999 }, { 0.316281835, 0.0534901321,
0.425116277 }, { 0.322609671, 0.0556335178, 0.426376869 }, { 0.328920763, 0.0578265505, 0.427510546 },
{ 0.335216916, 0.0600598734, 0.42852432 }, { 0.341499828, 0.0623252772, 0.429424503 }, { 0.347771086,
0.06461561, 0.430216765 }, { 0.354032169, 0.0669246832, 0.430906186 }, { 0.360284449, 0.0692471753,
0.431497309 }, { 0.366529195, 0.0715785403, 0.431994185 }, { 0.372767575, 0.0739149211, 0.432400419 },
{ 0.379000659, 0.0762530701, 0.432719214 }, { 0.385228383, 0.0785914864, 0.432954973 }, { 0.391452659,
0.0809267058, 0.433108763 }, { 0.397674379, 0.0832568129, 0.433182647 }, { 0.403894278, 0.0855803445,
0.433178526 }, { 0.410113015, 0.0878961593, 0.433098056 }, { 0.416331169, 0.0902033992, 0.432942678 },
{ 0.422549249, 0.0925014543, 0.432713635 }, { 0.428767696, 0.0947899342, 0.432411996 }, { 0.434986885,
0.0970686417, 0.432038673 }, { 0.441207124, 0.099337551, 0.431594438 }, { 0.447428382, 0.101597079,
0.431080497 }, { 0.453650614, 0.103847716, 0.430497898 }, { 0.459874623, 0.106089165, 0.429845789 }, {
0.466100494, 0.108321923, 0.429124507 }, { 0.472328255, 0.110546584, 0.42833432 }, { 0.478557889,
0.112763831, 0.427475431 }, { 0.484789325, 0.11497443, 0.426547991 }, { 0.491022448, 0.117179219,
0.425552106 }, { 0.497257069, 0.119379132, 0.424487908 }, { 0.503492698, 0.121575414, 0.42335611 }, {
```

```
0.509729541, 0.123768654, 0.422155676 }, { 0.515967304, 0.125959947, 0.420886594 }, { 0.522205646,
0.128150439, 0.419548848 }, { 0.528444192, 0.130341324, 0.418142411 }, { 0.534682523, 0.132533845,
0.416667258 }, { 0.540920186, 0.134729286, 0.415123366 }, { 0.547156706, 0.136928959, 0.413510662 }, {
0.553391649, 0.139134147, 0.411828882 }, { 0.559624442, 0.141346265, 0.410078028 }, { 0.565854477,
0.143566769, 0.408258132 }, { 0.572081108, 0.14579715, 0.406369246 }, { 0.578303656, 0.148038934,
0.404411444 }, { 0.584521407, 0.150293679, 0.402384829 }, { 0.590733615, 0.152562977, 0.400289528 }, {
0.596939751, 0.154848232, 0.398124897 }, { 0.60313893, 0.157151161, 0.395891308 }, { 0.609330184,
0.159473549, 0.393589349 }, { 0.615512627, 0.161817111, 0.391219295 }, { 0.62168534, 0.164183582,
0.388781456 }, { 0.627847374, 0.166574724, 0.38627618 }, { 0.633997746, 0.168992314, 0.383703854 }, {
0.640135447, 0.17143815, 0.381064906 }, { 0.646259648, 0.173913876, 0.378358969 }, { 0.652369348,
0.176421271, 0.375586209 }, { 0.658463166, 0.178962399, 0.372748214 }, { 0.664539964, 0.181539111,
0.369845599 }, { 0.670598572, 0.184153268, 0.366879025 }, { 0.676637795, 0.186806728, 0.363849195 }, {
0.682656407, 0.189501352, 0.360756856 }, { 0.688653158, 0.192238994, 0.357602797 }, { 0.694626769,
0.1950215, 0.354387853 }, { 0.700575937, 0.197850703, 0.3511129 }, { 0.706499709, 0.200728196,
0.347776863 }, { 0.712396345, 0.203656029, 0.344382594 }, { 0.718264447, 0.206635993, 0.340931208 }, {
0.724102613, 0.209669834, 0.337423766 }, { 0.729909422, 0.21275927, 0.333861367 }, { 0.735683432,
0.215905976, 0.330245147 }, { 0.741423185, 0.219111589, 0.326576275 }, { 0.747127207, 0.222377697,
0.322855952 }, { 0.752794009, 0.225705837, 0.31908541 }, { 0.75842209, 0.229097492, 0.31526591 }, {
0.76400994, 0.232554083, 0.311398734 }, { 0.769556038, 0.236076967, 0.307485188 }, { 0.775058888,
0.239667435, 0.303526312 }, { 0.780517023, 0.24332672, 0.299522665 }, { 0.785928794, 0.247055968,
0.295476756 }, { 0.791292674, 0.250856232, 0.291389943 }, { 0.796607144, 0.254728485, 0.287263585 }, {
0.801870689, 0.25867361, 0.283099033 }, { 0.807081807, 0.262692401, 0.278897629 }, { 0.812239008,
0.266785558, 0.274660698 }, { 0.817340818, 0.270953688, 0.270389545 }, { 0.822385784, 0.2751973,
0.266085445 }, { 0.827372474, 0.279516805, 0.261749643 }, { 0.832299481, 0.283912516, 0.257383341 }, {
0.837165425, 0.288384647, 0.2529877 }, { 0.841968959, 0.292933312, 0.248563825 }, { 0.846708768,
0.297558528, 0.244112767 }, { 0.851383572, 0.302260213, 0.239635512 }, { 0.85599213, 0.307038188,
0.235132978 }, { 0.860533241, 0.311892183, 0.230606009 }, { 0.865005747, 0.316821833, 0.226055368 }, {
0.869408534, 0.321826685, 0.221481734 }, { 0.87374053, 0.326906201, 0.216885699 }, { 0.878000715,
0.33205976, 0.212267762 }, { 0.882188112, 0.337286663, 0.207628326 }, { 0.886301795, 0.342586137,
0.202967696 }, { 0.890340885, 0.34795734, 0.19828608 }, { 0.894304553, 0.353399363, 0.193583583 }, {
0.898192017, 0.35891124, 0.188860212 }, { 0.902002544, 0.364491949, 0.184115876 }, { 0.905735448,
0.370140419, 0.179350388 }, { 0.90939009, 0.375855533, 0.174563472 }, { 0.912965874, 0.381636138,
0.169754764 }, { 0.916462251, 0.387481044, 0.164923826 }, { 0.91987871, 0.393389034, 0.160070152 }, {
0.923214783, 0.399358867, 0.155193185 }, { 0.926470039, 0.405389282, 0.150292329 }, { 0.929644083,
0.411479007, 0.145366973 }, { 0.932736555, 0.417626756, 0.140416519 }, { 0.935747126, 0.423831237,
0.135440416 }, { 0.938675494, 0.430091162, 0.130438175 }, { 0.941521384, 0.436405243, 0.12540944 }, {
0.944284543, 0.442772199, 0.120354038 }, { 0.946964741, 0.449190757, 0.115272059 }, { 0.949561766,
0.455659658, 0.110163947 }, { 0.952075421, 0.462177656, 0.105030614 }, { 0.954505523, 0.468743522,
0.0998735931 }, { 0.956851903, 0.475356048, 0.0946952268 }, { 0.959114397, 0.482014044, 0.0894989073
}, { 0.96129285, 0.488716345, 0.0842893891 }, { 0.96338711, 0.495461806, 0.0790731907 }, {
0.965397031, 0.502249309, 0.0738591143 }, { 0.967322465, 0.509077761, 0.0686589199 }, { 0.969163264,
0.515946092, 0.0634881971 }, { 0.970919277, 0.522853259, 0.058367489 }, { 0.972590351, 0.529798246,
0.0533237243 }, { 0.974176327, 0.536780059, 0.048392009 }, { 0.975677038, 0.543797733, 0.0436177922 },
{ 0.977092313, 0.550850323, 0.0390500131 }, { 0.978421971, 0.557936911, 0.0349306227 }, { 0.979665824,
0.5650566, 0.0314091591 }, { 0.980823673, 0.572208516, 0.0285075931 }, { 0.981895311, 0.579391803,
0.0262497353 }, { 0.982880522, 0.586605627, 0.0246613416 }, { 0.983779081, 0.593849168, 0.0237702263
}, { 0.984590755, 0.601121626, 0.0236063833 }, { 0.985315301, 0.608422211, 0.0242021174 }, {
0.985952471, 0.615750147, 0.0255921853 }, { 0.986502013, 0.623104667, 0.0278139496 }, { 0.98696367,
0.630485011, 0.0309075459 }, { 0.987337182, 0.637890424, 0.0349160639 }, { 0.987622296, 0.645320152,
0.0398857472 }, { 0.987818759, 0.652773439, 0.0455808037 }, { 0.98792633, 0.660249526, 0.0517503867 },
{ 0.987944783, 0.667747641, 0.0583286889 }, { 0.98787391, 0.675267, 0.0652570167 }, { 0.987713535,
0.682806802, 0.072489233 }, { 0.987463516, 0.690366218, 0.0799897176 }, { 0.987123759, 0.697944391,
0.0877314215 }, { 0.986694229, 0.705540424, 0.0956941797 }, { 0.98617497, 0.713153375, 0.103863324 },
{ 0.985565739, 0.72078246, 0.112228756 }, { 0.984865203, 0.728427497, 0.120784651 }, { 0.984075129,
0.736086521, 0.129526579 }, { 0.983195992, 0.743758326, 0.138453063 }, { 0.982228463, 0.751441596,
0.147564573 }, { 0.981173457, 0.759134892, 0.156863224 }, { 0.980032178, 0.766836624, 0.166352544 }, {
0.978806183, 0.774545028, 0.176037298 }, { 0.977497453, 0.782258138, 0.185923357 }, { 0.976108474,
0.789973753, 0.196017589 }, { 0.974637842, 0.797691563, 0.206331925 }, { 0.973087939, 0.805409333,
0.216876839 }, { 0.971467822, 0.813121725, 0.227658046 }, { 0.969783146, 0.820825143, 0.238685942 }, {
0.968040817, 0.828515491, 0.249971582 }, { 0.966242589, 0.836190976, 0.261533898 }, { 0.964393924,
0.843848069, 0.273391112 }, { 0.962516656, 0.85147634, 0.285545675 }, { 0.960625545, 0.859068716,
0.298010219 }, { 0.958720088, 0.866624355, 0.310820466 }, { 0.956834075, 0.874128569, 0.323973947 }, {
0.954997177, 0.881568926, 0.337475479 }, { 0.953215092, 0.888942277, 0.351368713 }, { 0.951546225,
0.896225909, 0.365627005 }, { 0.950018481, 0.903409063, 0.380271225 }, { 0.948683391, 0.910472964,
0.395289169 }, { 0.947594362, 0.917399053, 0.410665194 }, { 0.946809163, 0.924168246, 0.426373236 }, {
0.946391536, 0.930760752, 0.442367495 }, { 0.946402951, 0.937158971, 0.458591507 }, { 0.946902568,
0.943347775, 0.474969778 }, { 0.947936825, 0.949317522, 0.491426053 }, { 0.94954483, 0.9550629,
0.507859649 }, { 0.951740304, 0.960586693, 0.524203026 }, { 0.954529281, 0.965895868, 0.540360752 }, {
0.957896053, 0.97100333, 0.55627509 }, { 0.96181202, 0.975924241, 0.571925382 }, { 0.966248822,
0.980678193, 0.587205773 }, { 0.971161622, 0.985282161, 0.60215433 }, { 0.976510983, 0.989753437,
0.616760413 }, { 0.982257307, 0.994108844, 0.631017009 }, { 0.988362068, 0.998364143, 0.644924005 } };
00062         private static double[,] PlasmaRGB = new double[256, 3] { { 0.0503832136, 0.0298028976,
0.527974883 }, { 0.0635363639, 0.0284259729, 0.533123681 }, { 0.0753531234, 0.0272063728, 0.538007001
}, { 0.0862217979, 0.0261253206, 0.542657691 }, { 0.0963786097, 0.0251650976, 0.547103487 }, {
0.105979704, 0.0243092436, 0.551367851 }, { 0.115123641, 0.02355625, 0.555467728 }, { 0.123902903,
0.0228781011, 0.55942348 }, { 0.13238072, 0.0222583774, 0.563250116 }, { 0.140603076, 0.0216866674,
0.566959485 }, { 0.148606527, 0.0211535876, 0.570561711 }, { 0.156420649, 0.0206507174, 0.574065446 },
{ 0.164069722, 0.0201705326, 0.577478074 }, { 0.171573925, 0.0197063415, 0.58080589 }, { 0.178950212,
0.0192522243, 0.584054243 }, { 0.186212958, 0.0188029767, 0.587227661 }, { 0.193374449, 0.0183540593,
0.590329954 }, { 0.20044526, 0.0179015512, 0.593364304 }, { 0.207434551, 0.0174421086, 0.596333341 },
{ 0.214350298, 0.0169729276, 0.599239207 }, { 0.22119675, 0.0164970484, 0.602083323 }, { 0.227982971,
0.0160071509, 0.604867403 }, { 0.234714537, 0.0155015065, 0.607592438 }, { 0.241396253, 0.0149791041,
0.610259089 }, { 0.248032377, 0.0144393586, 0.612867743 }, { 0.25462669, 0.0138820918, 0.615418537 },
{ 0.261182562, 0.0133075156, 0.617911385 }, { 0.267702993, 0.0127162163, 0.620345997 }, { 0.274190665,
0.0121091423, 0.622721903 }, { 0.280647969, 0.0114875915, 0.625038468 }, { 0.287076059, 0.0108554862,
0.627294975 }, { 0.293477695, 0.0102128849, 0.62949049 }, { 0.299855122, 0.00956079551, 0.631623923 },
{ 0.306209825, 0.00890185346, 0.633694102 }, { 0.312543124, 0.00823900704, 0.635699759 }, {
```

```
0.318856183, 0.00757551051, 0.637639537 }, { 0.325150025, 0.00691491734, 0.639512001 }, { 0.331425547,
0.00626107379, 0.641315649 }, { 0.337683446, 0.00561830889, 0.643048936 }, { 0.343924591,
0.0049905308, 0.644710195 }, { 0.350149699, 0.00438202557, 0.646297711 }, { 0.356359209,
0.00379781761, 0.647809772 }, { 0.362553473, 0.00324319591, 0.649244641 }, { 0.368732762,
0.00272370721, 0.650600561 }, { 0.37489727, 0.00224514897, 0.651875762 }, { 0.381047116,
0.00181356205, 0.653068467 }, { 0.387182639, 0.00143446923, 0.654176761 }, { 0.39330401,
0.00111388259, 0.655198755 }, { 0.399410821, 0.000859420809, 0.656132835 }, { 0.405502914,
0.000678091517, 0.656977276 }, { 0.411580082, 0.000577101735, 0.65773038 }, { 0.417642063,
0.000563847476, 0.658390492 }, { 0.423688549, 0.00064590278, 0.658956004 }, { 0.429719186,
0.000831008207, 0.659425363 }, { 0.435733575, 0.00112705875, 0.659797077 }, { 0.441732123,
0.00153984779, 0.660069009 }, { 0.4477136, 0.00207954744, 0.660240367 }, { 0.453677394, 0.00275470302,
0.660309966 }, { 0.459622938, 0.00357374415, 0.660276655 }, { 0.465549631, 0.00454518084, 0.660139383
}, { 0.471456847, 0.00567758762, 0.65989721 }, { 0.477343929, 0.00697958743, 0.659549311 }, {
0.483210198, 0.00845983494, 0.659094989 }, { 0.489054951, 0.0101269996, 0.658533677 }, { 0.494877466,
0.0119897486, 0.657864946 }, { 0.500677687, 0.0140555064, 0.657087561 }, { 0.506454143, 0.0163333443,
0.656202294 }, { 0.512206035, 0.0188332232, 0.655209222 }, { 0.51793258, 0.0215631918, 0.654108545 },
{ 0.52363299, 0.0245316468, 0.652900629 }, { 0.529306474, 0.0277468735, 0.65158601 }, { 0.534952244,
0.03121703, 0.650165396 }, { 0.54056951, 0.034950131, 0.648639668 }, { 0.546157494, 0.0389540334,
0.647009884 }, { 0.551715423, 0.0431364795, 0.645277275 }, { 0.557242538, 0.0473307585, 0.64344325 },
{ 0.562738096, 0.0515448092, 0.641509389 }, { 0.568201372, 0.0557776706, 0.63947744 }, { 0.573631859,
0.0600281369, 0.637348841 }, { 0.579028682, 0.0642955547, 0.635126108 }, { 0.584391137, 0.0685790261,
0.632811608 }, { 0.589718606, 0.0728775875, 0.630407727 }, { 0.595010505, 0.0771902878, 0.627916992 },
{ 0.600266283, 0.0815161895, 0.625342058 }, { 0.605485428, 0.0858543713, 0.622685703 }, { 0.610667469,
0.0902039303, 0.619950811 }, { 0.615811974, 0.0945639838, 0.617140367 }, { 0.620918555, 0.0989336721,
0.61425744 }, { 0.625986869, 0.10331216, 0.611305174 }, { 0.631016615, 0.107698641, 0.608286774 }, {
0.636007543, 0.112092335, 0.605205491 }, { 0.640959444, 0.116492495, 0.602064611 }, { 0.645872158,
0.120898405, 0.598867442 }, { 0.650745571, 0.125309384, 0.5956173 }, { 0.655579615, 0.129724785,
0.592317494 }, { 0.660374266, 0.134143997, 0.588971318 }, { 0.665129493, 0.138566428, 0.585582301 }, {
0.669845385, 0.14299154, 0.582153572 }, { 0.67452206, 0.147418835, 0.578688247 }, { 0.679159664,
0.151847851, 0.575189431 }, { 0.683758384, 0.156278163, 0.571660158 }, { 0.68831844, 0.160709387,
0.56810338 }, { 0.692840088, 0.165141174, 0.564521958 }, { 0.697323615, 0.169573215, 0.560918659 }, {
0.701769334, 0.174005236, 0.557296144 }, { 0.70617759, 0.178437, 0.55365697 }, { 0.710548747,
0.182868306, 0.550003579 }, { 0.714883195, 0.187298986, 0.546338299 }, { 0.719181339, 0.191728906,
0.542663338 }, { 0.723443604, 0.196157962, 0.538980786 }, { 0.727670428, 0.200586086, 0.535292612 }, {
0.731862231, 0.205013174, 0.531600995 }, { 0.736019424, 0.209439071, 0.527908434 }, { 0.740142557,
0.213863965, 0.524215533 }, { 0.744232102, 0.218287899, 0.520523766 }, { 0.748288533, 0.222710942,
0.516834495 }, { 0.752312321, 0.227133187, 0.513148963 }, { 0.756303937, 0.231554749, 0.509468305 }, {
0.760263849, 0.235975765, 0.505793543 }, { 0.764192516, 0.240396394, 0.502125599 }, { 0.768090391,
0.244816813, 0.49846529 }, { 0.771957916, 0.24923722, 0.494813338 }, { 0.775795522, 0.253657797,
0.491170517 }, { 0.779603614, 0.258078397, 0.487539124 }, { 0.783382636, 0.262499662, 0.483917732 }, {
0.787132978, 0.266921859, 0.480306702 }, { 0.790855015, 0.271345267, 0.476706319 }, { 0.794549101,
0.275770179, 0.473116798 }, { 0.798215577, 0.280196901, 0.469538286 }, { 0.801854758, 0.28462575,
0.465970871 }, { 0.805466945, 0.289057057, 0.46241458 }, { 0.809052419, 0.293491117, 0.458869577 }, {
0.812611506, 0.297927865, 0.455337565 }, { 0.816144382, 0.30236813, 0.451816385 }, { 0.819651255,
0.306812282, 0.448305861 }, { 0.823132309, 0.311260703, 0.444805781 }, { 0.826587706, 0.315713782,
0.441315901 }, { 0.830017584, 0.320171913, 0.437835947 }, { 0.833422053, 0.324635499, 0.434365616 }, {
0.836801237, 0.329104836, 0.430905052 }, { 0.840155276, 0.333580106, 0.427454836 }, { 0.843484103,
0.338062109, 0.424013059 }, { 0.846787726, 0.342551272, 0.420579333 }, { 0.850066132, 0.347048028,
0.417153264 }, { 0.853319279, 0.351552815, 0.413734445 }, { 0.856547103, 0.356066072, 0.410322469 }, {
0.85974952, 0.360588229, 0.406916975 }, { 0.862926559, 0.365119408, 0.403518809 }, { 0.86607792,
0.369660446, 0.400126027 }, { 0.869203436, 0.374211795, 0.396738211 }, { 0.872302917, 0.37877391,
0.393354947 }, { 0.875376149, 0.383347243, 0.389975832 }, { 0.878422895, 0.387932249, 0.386600468 }, {
0.881442916, 0.392529339, 0.383228622 }, { 0.884435982, 0.397138877, 0.379860246 }, { 0.887401682,
0.401761511, 0.376494232 }, { 0.890339687, 0.406397694, 0.373130228 }, { 0.893249647, 0.411047871,
0.369767893 }, { 0.896131191, 0.415712489, 0.366406907 }, { 0.898983931, 0.420391986, 0.363046965 }, {
0.901807455, 0.425086807, 0.359687758 }, { 0.904601295, 0.429797442, 0.356328796 }, { 0.907364995,
0.434524335, 0.352969777 }, { 0.910098088, 0.439267908, 0.349610469 }, { 0.912800095, 0.444028574,
0.346250656 }, { 0.915470518, 0.448806744, 0.342890148 }, { 0.918108848, 0.453602818, 0.339528771 }, {
0.920714383, 0.45841742, 0.336165582 }, { 0.92328666, 0.463250828, 0.332800827 }, { 0.925825146,
0.468103387, 0.329434512 }, { 0.928329275, 0.472975465, 0.32606655 }, { 0.930798469, 0.47786742,
0.322696876 }, { 0.93323214, 0.482779603, 0.319325444 }, { 0.935629684, 0.487712357, 0.315952211 }, {
0.937990034, 0.492666544, 0.31257544 }, { 0.940312939, 0.497642038, 0.309196628 }, { 0.942597771,
0.502639147, 0.305815824 }, { 0.944843893, 0.507658169, 0.302433101 }, { 0.947050662, 0.51269939,
0.299048555 }, { 0.949217427, 0.517763087, 0.295662308 }, { 0.95134353, 0.522849522, 0.292274506 }, {
0.953427725, 0.52795955, 0.288883445 }, { 0.95546964, 0.533093083, 0.285490391 }, { 0.95746877,
0.538250172, 0.282096149 }, { 0.95942443, 0.543431038, 0.27870099 }, { 0.96133593, 0.54863589,
0.275305214 }, { 0.963202573, 0.553864931, 0.271909159 }, { 0.965023656, 0.559118349, 0.2685132 }, {
0.96679847, 0.564396327, 0.265117752 }, { 0.968525639, 0.569699633, 0.261721488 }, { 0.970204593,
0.57502827, 0.258325424 }, { 0.971835007, 0.580382015, 0.254931256 }, { 0.973416145, 0.585761012,
0.251539615 }, { 0.974947262, 0.591165394, 0.2481512 }, { 0.976427606, 0.596595287, 0.244766775 }, {
0.977856416, 0.602050811, 0.241387186 }, { 0.979232922, 0.607532077, 0.238013359 }, { 0.980556344,
0.61303919, 0.234646316 }, { 0.98182589, 0.61857225, 0.231287178 }, { 0.983040742, 0.624131362,
0.227937141 }, { 0.984198924, 0.629717516, 0.224595006 }, { 0.98530076, 0.635329876, 0.221264889 }, {
0.986345421, 0.640968508, 0.217948456 }, { 0.987332067, 0.646633475, 0.214647532 }, { 0.988259846,
0.652324832, 0.211364122 }, { 0.989127893, 0.65804263, 0.208100426 }, { 0.989935328, 0.663786914,
0.204858855 }, { 0.990681261, 0.66955772, 0.201642049 }, { 0.991364787, 0.675355082, 0.1984529 }, {
0.99198499, 0.681179025, 0.195294567 }, { 0.992540939, 0.687029567, 0.1921705 }, { 0.993031693,
0.692906719, 0.189084459 }, { 0.993456302, 0.698810484, 0.186040537 }, { 0.993813802, 0.704740854,
0.18304318 }, { 0.994103226, 0.710697814, 0.180097207 }, { 0.994323596, 0.716681336, 0.177207826 }, {
0.994473934, 0.722691379, 0.174380656 }, { 0.99455326, 0.72872789, 0.171621733 }, { 0.994560594,
0.734790799, 0.168937522 }, { 0.994494964, 0.74088002, 0.166334918 }, { 0.994355411, 0.746995448,
0.163821243 }, { 0.994140989, 0.753136955, 0.161404226 }, { 0.993850778, 0.75930439, 0.159091984 }, {
0.99348219, 0.765498551, 0.156890625 }, { 0.993033251, 0.771719833, 0.154807583 }, { 0.992505214,
0.777966775, 0.152854862 }, { 0.99189727, 0.78423912, 0.151041581 }, { 0.99120868, 0.790536569,
0.149376885 }, { 0.990438793, 0.796858775, 0.14786981 }, { 0.989587065, 0.803205337, 0.146529128 }, {
0.988647741, 0.809578605, 0.145357284 }, { 0.987620557, 0.815977942, 0.144362644 }, { 0.986509366,
0.82240062, 0.143556679 }, { 0.985314198, 0.82884598, 0.142945116 }, { 0.984031139, 0.83531536,
```

```
0.142528388 }, { 0.98265282, 0.84181173, 0.142302653 }, { 0.981190389, 0.848328902, 0.142278607 }, {
0.979643637, 0.854866468, 0.142453425 }, { 0.977994918, 0.861432314, 0.142808191 }, { 0.976264977,
0.868015998, 0.143350944 }, { 0.974443038, 0.874622194, 0.144061156 }, { 0.972530009, 0.881250063,
0.144922913 }, { 0.970532932, 0.887896125, 0.145918663 }, { 0.968443477, 0.894563989, 0.147014438 }, {
0.966271225, 0.901249365, 0.148179639 }, { 0.964021057, 0.907950379, 0.149370428 }, { 0.961681481,
0.914672479, 0.150520343 }, { 0.959275646, 0.921406537, 0.151566019 }, { 0.956808068, 0.928152065,
0.152409489 }, { 0.954286813, 0.93490773, 0.152921158 }, { 0.951726083, 0.941670605, 0.152925363 }, {
0.949150533, 0.9484349, 0.152177604 }, { 0.94660227, 0.95518986, 0.150327944 }, { 0.944151742,
0.961916487, 0.146860789 }, { 0.94189612, 0.968589814, 0.140955606 }, { 0.940015097, 0.975158357,
0.131325517 } };
00063         private static double[,] ViridisRGB = new double[256, 3] { { 0.26700401, 0.00487433,
0.32941519 }, { 0.26851048, 0.00960483, 0.33542652 }, { 0.26994384, 0.01462494, 0.34137895 }, {
0.27130489, 0.01994186, 0.34726862 }, { 0.27259384, 0.02556309, 0.35309303 }, { 0.27380934,
0.03149748, 0.35885256 }, { 0.27495242, 0.03775181, 0.36454323 }, { 0.27602238, 0.04416723, 0.37016418
}, { 0.2770184, 0.05034437, 0.37571452 }, { 0.27794143, 0.05632444, 0.38119074 }, { 0.27879067,
0.06214536, 0.38659204 }, { 0.2795655, 0.06783587, 0.39191723 }, { 0.28026658, 0.07341724, 0.39716349
}, { 0.28089358, 0.07890703, 0.40232944 }, { 0.28144581, 0.0843197, 0.40741404 }, { 0.28192358,
0.08966622, 0.41241521 }, { 0.28232739, 0.09495545, 0.41733086 }, { 0.28265633, 0.10019576, 0.42216032
}, { 0.28291049, 0.10539345, 0.42690202 }, { 0.28309095, 0.11055307, 0.43155375 }, { 0.28319704,
0.11567966, 0.43611482 }, { 0.28322882, 0.12077701, 0.44058404 }, { 0.28318684, 0.12584799, 0.44496 },
{ 0.283072, 0.13089477, 0.44924127 }, { 0.28288389, 0.13592005, 0.45342734 }, { 0.28262297,
0.14092556, 0.45751726 }, { 0.28229037, 0.14591233, 0.46150995 }, { 0.28188676, 0.15088147, 0.46540474
}, { 0.28141228, 0.15583425, 0.46920128 }, { 0.28086773, 0.16077132, 0.47289909 }, { 0.28025468,
0.16569272, 0.47649762 }, { 0.27957399, 0.17059884, 0.47999675 }, { 0.27882618, 0.1754902, 0.48339654
}, { 0.27801236, 0.18036684, 0.48669702 }, { 0.27713437, 0.18522836, 0.48989831 }, { 0.27619376,
0.19007447, 0.49300074 }, { 0.27519116, 0.1949054, 0.49600488 }, { 0.27412802, 0.19972086, 0.49891131
}, { 0.27300596, 0.20452049, 0.50172076 }, { 0.27182812, 0.20930306, 0.50443413 }, { 0.27059473,
0.21406899, 0.50705243 }, { 0.26930756, 0.21881782, 0.50957678 }, { 0.26796846, 0.22354911, 0.5120084
}, { 0.26657984, 0.2282621, 0.5143487 }, { 0.2651445, 0.23295593, 0.5165993 }, { 0.2636632,
0.23763078, 0.51876163 }, { 0.26213801, 0.24228619, 0.52083736 }, { 0.26057103, 0.2469217, 0.52282822
}, { 0.25896451, 0.25153685, 0.52473609 }, { 0.25732244, 0.2561304, 0.52656332 }, { 0.25564519,
0.26070284, 0.52831152 }, { 0.25393498, 0.26525384, 0.52998273 }, { 0.25219404, 0.26978306, 0.53157905
}, { 0.25042462, 0.27429024, 0.53310261 }, { 0.24862899, 0.27877509, 0.53455561 }, { 0.2468114,
0.28323662, 0.53594093 }, { 0.24497208, 0.28767547, 0.53726018 }, { 0.24311324, 0.29209154, 0.53851561
}, { 0.24123708, 0.29648471, 0.53970946 }, { 0.23934575, 0.30085494, 0.54084398 }, { 0.23744138,
0.30520222, 0.5419214 }, { 0.23552606, 0.30952657, 0.54294396 }, { 0.23360277, 0.31382773, 0.54391424
}, { 0.2316735, 0.3181058, 0.54483444 }, { 0.22973926, 0.32236127, 0.54570633 }, { 0.22780192,
0.32659432, 0.546532 }, { 0.2258633, 0.33080515, 0.54731353 }, { 0.22392515, 0.334994, 0.54805291 }, {
0.22198915, 0.33916114, 0.54875211 }, { 0.22005691, 0.34330688, 0.54941304 }, { 0.21812995,
0.34743154, 0.55003755 }, { 0.21620971, 0.35153548, 0.55062743 }, { 0.21429757, 0.35561907, 0.5511844
}, { 0.21239477, 0.35968273, 0.55171011 }, { 0.2105031, 0.36372671, 0.55220646 }, { 0.20862342,
0.36775151, 0.55267486 }, { 0.20675628, 0.37175775, 0.55311653 }, { 0.20490257, 0.37574589, 0.55353282
}, { 0.20306309, 0.37971644, 0.55392505 }, { 0.20123854, 0.38366989, 0.55429441 }, { 0.1994295,
0.38760678, 0.55464205 }, { 0.1976365, 0.39152762, 0.55496905 }, { 0.19585993, 0.39543297, 0.55527637
}, { 0.19410009, 0.39932336, 0.55556494 }, { 0.19235719, 0.40319934, 0.55583559 }, { 0.19063135,
0.40706148, 0.55608907 }, { 0.18892259, 0.41091033, 0.55632606 }, { 0.18723083, 0.41474645, 0.55654717
}, { 0.18555593, 0.4185704, 0.55675292 }, { 0.18389763, 0.42238275, 0.55694377 }, { 0.18225561,
0.42618405, 0.5571201 }, { 0.18062949, 0.42997486, 0.55728221 }, { 0.17901879, 0.43375572, 0.55743035
}, { 0.17742298, 0.4375272, 0.55756466 }, { 0.17584148, 0.44128981, 0.55768526 }, { 0.17427363,
0.4450441, 0.55779216 }, { 0.17271876, 0.4487906, 0.55788532 }, { 0.17117615, 0.4525298, 0.55796464 },
{ 0.16964573, 0.45626209, 0.55803034 }, { 0.16812641, 0.45998802, 0.55808199 }, { 0.1666171,
0.46370813, 0.55811913 }, { 0.16511703, 0.4674229, 0.55814141 }, { 0.16362543, 0.47113278, 0.55814842
}, { 0.16214155, 0.47483821, 0.55813967 }, { 0.16066467, 0.47853961, 0.55811466 }, { 0.15919413,
0.4822374, 0.5580728 }, { 0.15772933, 0.48593197, 0.55801347 }, { 0.15626973, 0.4896237, 0.557936 }, {
0.15481488, 0.49331293, 0.55783967 }, { 0.15336445, 0.49700003, 0.55772371 }, { 0.1519182, 0.50068529,
0.55758733 }, { 0.15047605, 0.50436904, 0.55742968 }, { 0.14903918, 0.50805136, 0.5572505 }, {
0.14760731, 0.51173263, 0.55704861 }, { 0.14618026, 0.51541316, 0.55682271 }, { 0.14475863,
0.51909319, 0.55657181 }, { 0.14334327, 0.52277292, 0.55629491 }, { 0.14193527, 0.52645254, 0.55599097
}, { 0.14053599, 0.53013219, 0.55565893 }, { 0.13914708, 0.53381201, 0.55529773 }, { 0.13777048,
0.53749213, 0.55490625 }, { 0.1364085, 0.54117264, 0.55448339 }, { 0.13506561, 0.54485335, 0.55402906
}, { 0.13374299, 0.54853458, 0.55354108 }, { 0.13244401, 0.55221637, 0.55301828 }, { 0.13117249,
0.55589872, 0.55245948 }, { 0.1299327, 0.55958162, 0.55186354 }, { 0.12872938, 0.56326503, 0.55122927
}, { 0.12756771, 0.56694891, 0.55055551 }, { 0.12645338, 0.57063316, 0.5498411 }, { 0.12539383,
0.57431754, 0.54908564 }, { 0.12439474, 0.57800205, 0.5482874 }, { 0.12346281, 0.58168661, 0.54744498
}, { 0.12260562, 0.58537105, 0.54655722 }, { 0.12183122, 0.58905521, 0.54562298 }, { 0.12114807,
0.59273889, 0.54464114 }, { 0.12056501, 0.59642187, 0.54361058 }, { 0.12009154, 0.60010387, 0.54253043
}, { 0.11973756, 0.60378459, 0.54139999 }, { 0.11951163, 0.60746388, 0.54021751 }, { 0.11942341,
0.61114146, 0.53898192 }, { 0.11948255, 0.61481702, 0.53769219 }, { 0.11969858, 0.61849025, 0.53634733
}, { 0.12008079, 0.62216081, 0.53494633 }, { 0.12063824, 0.62582833, 0.53348834 }, { 0.12137972,
0.62949242, 0.53197275 }, { 0.12231244, 0.63315277, 0.53039808 }, { 0.12344358, 0.63680899, 0.52876343
}, { 0.12477953, 0.64046069, 0.52706792 }, { 0.12632581, 0.64410744, 0.52531069 }, { 0.12808703,
0.64774881, 0.52349092 }, { 0.13006688, 0.65138436, 0.52160791 }, { 0.13226797, 0.65501363, 0.51966086
}, { 0.13469183, 0.65863619, 0.5176488 }, { 0.13733921, 0.66225157, 0.51557101 }, { 0.14020991,
0.66585927, 0.5134268 }, { 0.14330291, 0.66945881, 0.51121549 }, { 0.1466164, 0.67304968, 0.50893644
}, { 0.15014782, 0.67663139, 0.5065889 }, { 0.15389405, 0.68020343, 0.50417217 }, { 0.15785146,
0.68376525, 0.50168574 }, { 0.16201598, 0.68731632, 0.49912906 }, { 0.1663832, 0.69085611, 0.49650163
}, { 0.1709484, 0.69438405, 0.49380294 }, { 0.17570671, 0.6978996, 0.49103252 }, { 0.18065314,
0.70140222, 0.48818938 }, { 0.18578266, 0.70489133, 0.48527326 }, { 0.19109018, 0.70836635, 0.48228395
}, { 0.19657063, 0.71182668, 0.47922108 }, { 0.20221902, 0.71527175, 0.47608431 }, { 0.20803045,
0.71870095, 0.4728733 }, { 0.21400015, 0.72211371, 0.46958774 }, { 0.22012381, 0.72550945, 0.46622638
}, { 0.2263969, 0.72888753, 0.46278934 }, { 0.23281498, 0.73224735, 0.45927675 }, { 0.2393739,
0.73558828, 0.45568838 }, { 0.24606968, 0.73890972, 0.45202405 }, { 0.25289851, 0.74221104, 0.44828355
}, { 0.25985676, 0.74549162, 0.44446673 }, { 0.26694127, 0.74875084, 0.44057284 }, { 0.27414922,
0.75198807, 0.4366009 }, { 0.28147681, 0.75520266, 0.43255207 }, { 0.28892102, 0.75839399, 0.42842626
}, { 0.29647899, 0.76156142, 0.42422341 }, { 0.30414796, 0.76470433, 0.41994346 }, { 0.31192534,
0.76782207, 0.41558638 }, { 0.3198086, 0.77091403, 0.41115215 }, { 0.3277958, 0.77397953, 0.40664011
}, { 0.33588539, 0.7770179, 0.40204917 }, { 0.34407411, 0.78002855, 0.39738103 }, { 0.35235985,
```

```
0.78301086, 0.39263579 }, { 0.36074053, 0.78596419, 0.38781353 }, { 0.3692142, 0.78888793, 0.38291438
}, { 0.37777892, 0.79178146, 0.3779385 }, { 0.38643282, 0.79464415, 0.37288606 }, { 0.39517408,
0.79747541, 0.36775726 }, { 0.40400101, 0.80027461, 0.36255223 }, { 0.4129135, 0.80304099, 0.35726893
}, { 0.42190813, 0.80577412, 0.35191009 }, { 0.43098317, 0.80847343, 0.34647607 }, { 0.44013691,
0.81113836, 0.3409673 }, { 0.44936763, 0.81376835, 0.33538426 }, { 0.45867362, 0.81636288, 0.32972749
}, { 0.46805314, 0.81892143, 0.32399761 }, { 0.47750446, 0.82144351, 0.31819529 }, { 0.4870258,
0.82392862, 0.31232133 }, { 0.49661536, 0.82637633, 0.30637661 }, { 0.5062713, 0.82878621, 0.30036211
}, { 0.51599182, 0.83115784, 0.29427888 }, { 0.52577622, 0.83349064, 0.2881265 }, { 0.5356211,
0.83578452, 0.28190832 }, { 0.5455244, 0.83803918, 0.27562602 }, { 0.55548397, 0.84025437, 0.26928147
}, { 0.5654976, 0.8424299, 0.26287683 }, { 0.57556297, 0.84456561, 0.25641457 }, { 0.58567772,
0.84666139, 0.24989748 }, { 0.59583934, 0.84871722, 0.24332878 }, { 0.60604528, 0.8507331, 0.23671214
}, { 0.61629283, 0.85270912, 0.23005179 }, { 0.62657923, 0.85464543, 0.22335258 }, { 0.63690157,
0.85654226, 0.21662012 }, { 0.64725685, 0.85839991, 0.20986086 }, { 0.65764197, 0.86021878, 0.20308229
}, { 0.66805369, 0.86199932, 0.19629307 }, { 0.67848868, 0.86374211, 0.18950326 }, { 0.68894351,
0.86544779, 0.18272455 }, { 0.69941463, 0.86711711, 0.17597055 }, { 0.70989842, 0.86875092, 0.16925712
}, { 0.72039115, 0.87035015, 0.16260273 }, { 0.73088902, 0.87191584, 0.15602894 }, { 0.74138803,
0.87344918, 0.14956101 }, { 0.75188414, 0.87495143, 0.14322828 }, { 0.76237342, 0.87642392, 0.13706449
}, { 0.77285183, 0.87786808, 0.13110864 }, { 0.78331535, 0.87928545, 0.12540538 }, { 0.79375994,
0.88067763, 0.12000532 }, { 0.80418159, 0.88204632, 0.11496505 }, { 0.81457634, 0.88339329, 0.11034678
}, { 0.82494028, 0.88472036, 0.10621724 }, { 0.83526959, 0.88602943, 0.1026459 }, { 0.84556056,
0.88732243, 0.09970219 }, { 0.8558096, 0.88860134, 0.09745186 }, { 0.86601325, 0.88986815, 0.09595277
}, { 0.87616824, 0.89112487, 0.09525046 }, { 0.88627146, 0.89237353, 0.09537439 }, { 0.89632002,
0.89361614, 0.09633538 }, { 0.90631121, 0.89485467, 0.09812496 }, { 0.91624212, 0.89609127, 0.1007168
}, { 0.92610579, 0.89732977, 0.10407067 }, { 0.93590444, 0.8985704, 0.10813094 }, { 0.94563626,
0.899815, 0.11283773 }, { 0.95529972, 0.90106534, 0.11812832 }, { 0.96489353, 0.90232311, 0.12394051
}, { 0.97441665, 0.90358991, 0.13021494 }, { 0.98386829, 0.90486726, 0.13689671 }, { 0.99324789,
0.90615657, 0.1439362 } };
00064         private static double[,] CividisRGB = new double[256, 3] { { 0, 0.1262, 0.3015 }, { 0, 0.1292,
0.3077 }, { 0, 0.1321, 0.3142 }, { 0, 0.135, 0.3205 }, { 0, 0.1379, 0.3269 }, { 0, 0.1408, 0.3334 }, {
0, 0.1437, 0.34 }, { 0, 0.1465, 0.3467 }, { 0, 0.1492, 0.3537 }, { 0, 0.1519, 0.3606 }, { 0, 0.1546,
0.3676 }, { 0, 0.1574, 0.3746 }, { 0, 0.1601, 0.3817 }, { 0, 0.1629, 0.3888 }, { 0, 0.1657, 0.396 }, {
0, 0.1685, 0.4031 }, { 0, 0.1714, 0.4102 }, { 0, 0.1743, 0.4172 }, { 0, 0.1773, 0.4241 }, { 0, 0.1798,
0.4307 }, { 0, 0.1817, 0.4347 }, { 0, 0.1834, 0.4363 }, { 0, 0.1852, 0.4368 }, { 0, 0.1872, 0.4368 },
{ 0, 0.1901, 0.4365 }, { 0, 0.193, 0.4361 }, { 0, 0.1958, 0.4356 }, { 0, 0.1987, 0.4349 }, { 0,
0.2015, 0.4343 }, { 0, 0.2044, 0.4336 }, { 0, 0.2073, 0.4329 }, { 0.0055, 0.2101, 0.4322 }, { 0.0236,
0.213, 0.4314 }, { 0.0416, 0.2158, 0.4308 }, { 0.0576, 0.2187, 0.4301 }, { 0.071, 0.2215, 0.4293 }, {
0.0827, 0.2244, 0.4287 }, { 0.0932, 0.2272, 0.428 }, { 0.103, 0.23, 0.4274 }, { 0.112, 0.2329, 0.4268
}, { 0.1204, 0.2357, 0.4262 }, { 0.1283, 0.2385, 0.4256 }, { 0.1359, 0.2414, 0.4251 }, { 0.1431,
0.2442, 0.4245 }, { 0.15, 0.247, 0.4241 }, { 0.1566, 0.2498, 0.4236 }, { 0.163, 0.2526, 0.4232 }, {
0.1692, 0.2555, 0.4228 }, { 0.1752, 0.2583, 0.4224 }, { 0.1811, 0.2611, 0.422 }, { 0.1868, 0.2639,
0.4217 }, { 0.1923, 0.2667, 0.4214 }, { 0.1977, 0.2695, 0.4212 }, { 0.203, 0.2723, 0.4209 }, { 0.2082,
0.2751, 0.4207 }, { 0.2133, 0.278, 0.4205 }, { 0.2183, 0.2808, 0.4204 }, { 0.2232, 0.2836, 0.4203 }, {
0.2281, 0.2864, 0.4202 }, { 0.2328, 0.2892, 0.4201 }, { 0.2375, 0.292, 0.42 }, { 0.2421, 0.2948, 0.42
}, { 0.2466, 0.2976, 0.42 }, { 0.2511, 0.3004, 0.4201 }, { 0.2556, 0.3032, 0.4201 }, { 0.2599, 0.306,
0.4202 }, { 0.2643, 0.3088, 0.4203 }, { 0.2686, 0.3116, 0.4205 }, { 0.2728, 0.3144, 0.4206 }, { 0.277,
0.3172, 0.4208 }, { 0.2811, 0.32, 0.421 }, { 0.2853, 0.3228, 0.4212 }, { 0.2894, 0.3256, 0.4215 }, {
0.2934, 0.3284, 0.4218 }, { 0.2974, 0.3312, 0.4221 }, { 0.3014, 0.334, 0.4224 }, { 0.3054, 0.3368,
0.4227 }, { 0.3093, 0.3396, 0.4231 }, { 0.3132, 0.3424, 0.4236 }, { 0.317, 0.3453, 0.424 }, { 0.3209,
0.3481, 0.4244 }, { 0.3247, 0.3509, 0.4249 }, { 0.3285, 0.3537, 0.4254 }, { 0.3323, 0.3565, 0.4259 },
{ 0.3361, 0.3593, 0.4264 }, { 0.3398, 0.3622, 0.427 }, { 0.3435, 0.365, 0.4276 }, { 0.3472, 0.3678,
0.4282 }, { 0.3509, 0.3706, 0.4288 }, { 0.3546, 0.3734, 0.4294 }, { 0.3582, 0.3763, 0.4302 }, {
0.3619, 0.3791, 0.4308 }, { 0.3655, 0.3819, 0.4316 }, { 0.3691, 0.3848, 0.4322 }, { 0.3727, 0.3876,
0.4331 }, { 0.3763, 0.3904, 0.4338 }, { 0.3798, 0.3933, 0.4346 }, { 0.3834, 0.3961, 0.4355 }, {
0.3869, 0.399, 0.4364 }, { 0.3905, 0.4018, 0.4372 }, { 0.394, 0.4047, 0.4381 }, { 0.3975, 0.4075,
0.439 }, { 0.401, 0.4104, 0.44 }, { 0.4045, 0.4132, 0.4409 }, { 0.408, 0.4161, 0.4419 }, { 0.4114,
0.4189, 0.443 }, { 0.4149, 0.4218, 0.444 }, { 0.4183, 0.4247, 0.445 }, { 0.4218, 0.4275, 0.4462 }, {
0.4252, 0.4304, 0.4473 }, { 0.4286, 0.4333, 0.4485 }, { 0.432, 0.4362, 0.4496 }, { 0.4354, 0.439,
0.4508 }, { 0.4388, 0.4419, 0.4521 }, { 0.4422, 0.4448, 0.4534 }, { 0.4456, 0.4477, 0.4547 }, {
0.4489, 0.4506, 0.4561 }, { 0.4523, 0.4535, 0.4575 }, { 0.4556, 0.4564, 0.4589 }, { 0.4589, 0.4593,
0.4604 }, { 0.4622, 0.4622, 0.462 }, { 0.4656, 0.4651, 0.4635 }, { 0.4689, 0.468, 0.465 }, { 0.4722,
0.4709, 0.4665 }, { 0.4756, 0.4738, 0.4679 }, { 0.479, 0.4767, 0.4691 }, { 0.4825, 0.4797, 0.4701 }, {
0.4861, 0.4826, 0.4707 }, { 0.4897, 0.4856, 0.4714 }, { 0.4934, 0.4886, 0.4719 }, { 0.4971, 0.4915,
0.4723 }, { 0.5008, 0.4945, 0.4727 }, { 0.5045, 0.4975, 0.473 }, { 0.5083, 0.5005, 0.4732 }, { 0.5121,
0.5035, 0.4734 }, { 0.5158, 0.5065, 0.4736 }, { 0.5196, 0.5095, 0.4737 }, { 0.5234, 0.5125, 0.4738 },
{ 0.5272, 0.5155, 0.4739 }, { 0.531, 0.5186, 0.4739 }, { 0.5349, 0.5216, 0.4738 }, { 0.5387, 0.5246,
0.4739 }, { 0.5425, 0.5277, 0.4738 }, { 0.5464, 0.5307, 0.4736 }, { 0.5502, 0.5338, 0.4735 }, {
0.5541, 0.5368, 0.4733 }, { 0.5579, 0.5399, 0.4732 }, { 0.5618, 0.543, 0.4729 }, { 0.5657, 0.5461,
0.4727 }, { 0.5696, 0.5491, 0.4723 }, { 0.5735, 0.5522, 0.472 }, { 0.5774, 0.5553, 0.4717 }, { 0.5813,
0.5584, 0.4714 }, { 0.5852, 0.5615, 0.4709 }, { 0.5892, 0.5646, 0.4705 }, { 0.5931, 0.5678, 0.4701 },
{ 0.597, 0.5709, 0.4696 }, { 0.601, 0.574, 0.4691 }, { 0.605, 0.5772, 0.4685 }, { 0.6089, 0.5803,
0.468 }, { 0.6129, 0.5835, 0.4673 }, { 0.6168, 0.5866, 0.4668 }, { 0.6208, 0.5898, 0.4662 }, { 0.6248,
0.5929, 0.4655 }, { 0.6288, 0.5961, 0.4649 }, { 0.6328, 0.5993, 0.4641 }, { 0.6368, 0.6025, 0.4632 },
{ 0.6408, 0.6057, 0.4625 }, { 0.6449, 0.6089, 0.4617 }, { 0.6489, 0.6121, 0.4609 }, { 0.6529, 0.6153,
0.46 }, { 0.657, 0.6185, 0.4591 }, { 0.661, 0.6217, 0.4583 }, { 0.6651, 0.625, 0.4573 }, { 0.6691,
0.6282, 0.4562 }, { 0.6732, 0.6315, 0.4553 }, { 0.6773, 0.6347, 0.4543 }, { 0.6813, 0.638, 0.4532 }, {
0.6854, 0.6412, 0.4521 }, { 0.6895, 0.6445, 0.4511 }, { 0.6936, 0.6478, 0.4499 }, { 0.6977, 0.6511,
0.4487 }, { 0.7018, 0.6544, 0.4475 }, { 0.706, 0.6577, 0.4463 }, { 0.7101, 0.661, 0.445 }, { 0.7142,
0.6643, 0.4437 }, { 0.7184, 0.6676, 0.4424 }, { 0.7225, 0.671, 0.4409 }, { 0.7267, 0.6743, 0.4396 }, {
0.7308, 0.6776, 0.4382 }, { 0.735, 0.681, 0.4368 }, { 0.7392, 0.6844, 0.4352 }, { 0.7434, 0.6877,
0.4338 }, { 0.7476, 0.6911, 0.4322 }, { 0.7518, 0.6945, 0.4307 }, { 0.756, 0.6979, 0.429 }, { 0.7602,
0.7013, 0.4273 }, { 0.7644, 0.7047, 0.4258 }, { 0.7686, 0.7081, 0.4241 }, { 0.7729, 0.7115, 0.4223 },
{ 0.7771, 0.715, 0.4205 }, { 0.7814, 0.7184, 0.4188 }, { 0.7856, 0.7218, 0.4168 }, { 0.7899, 0.7253,
0.415 }, { 0.7942, 0.7288, 0.4129 }, { 0.7985, 0.7322, 0.4111 }, { 0.8027, 0.7357, 0.409 }, { 0.807,
0.7392, 0.407 }, { 0.8114, 0.7427, 0.4049 }, { 0.8157, 0.7462, 0.4028 }, { 0.82, 0.7497, 0.4007 }, {
0.8243, 0.7532, 0.3984 }, { 0.8287, 0.7568, 0.3961 }, { 0.833, 0.7603, 0.3938 }, { 0.8374, 0.7639,
0.3915 }, { 0.8417, 0.7674, 0.3892 }, { 0.8461, 0.771, 0.3869 }, { 0.8505, 0.7745, 0.3843 }, { 0.8548,
0.7781, 0.3818 }, { 0.8592, 0.7817, 0.3793 }, { 0.8636, 0.7853, 0.3766 }, { 0.8681, 0.7889, 0.3739 },
```

```
{ 0.8725, 0.7926, 0.3712 }, { 0.8769, 0.7962, 0.3684 }, { 0.8813, 0.7998, 0.3657 }, { 0.8858, 0.8035,
0.3627 }, { 0.8902, 0.8071, 0.3599 }, { 0.8947, 0.8108, 0.3569 }, { 0.8992, 0.8145, 0.3538 }, {
0.9037, 0.8182, 0.3507 }, { 0.9082, 0.8219, 0.3474 }, { 0.9127, 0.8256, 0.3442 }, { 0.9172, 0.8293,
0.3409 }, { 0.9217, 0.833, 0.3374 }, { 0.9262, 0.8367, 0.334 }, { 0.9308, 0.8405, 0.3306 }, { 0.9353,
0.8442, 0.3268 }, { 0.9399, 0.848, 0.3232 }, { 0.9444, 0.8518, 0.3195 }, { 0.949, 0.8556, 0.3155 }, {
0.9536, 0.8593, 0.3116 }, { 0.9582, 0.8632, 0.3076 }, { 0.9628, 0.867, 0.3034 }, { 0.9674, 0.8708,
0.299 }, { 0.9721, 0.8746, 0.2947 }, { 0.9767, 0.8785, 0.2901 }, { 0.9814, 0.8823, 0.2856 }, { 0.986,
0.8862, 0.2807 }, { 0.9907, 0.8901, 0.2759 }, { 0.9954, 0.894, 0.2708 }, { 1, 0.8979, 0.2655 }, { 1,
0.9018, 0.26 }, { 1, 0.9057, 0.2593 }, { 1, 0.9094, 0.2634 }, { 1, 0.9131, 0.268 }, { 1, 0.9169,
0.2731 } };
00065         private static double[,] RocketRGB = new double[256, 3] { { 0.01060815, 0.01808215, 0.10018654
}, { 0.01428972, 0.02048237, 0.10374486 }, { 0.01831941, 0.0229766, 0.10738511 }, { 0.02275049,
0.02554464, 0.11108639 }, { 0.02759119, 0.02818316, 0.11483751 }, { 0.03285175, 0.03088792, 0.11863035
}, { 0.03853466, 0.03365771, 0.12245873 }, { 0.04447016, 0.03648425, 0.12631831 }, { 0.05032105,
0.03936808, 0.13020508 }, { 0.05611171, 0.04224835, 0.13411624 }, { 0.0618531, 0.04504866, 0.13804929
}, { 0.06755457, 0.04778179, 0.14200206 }, { 0.0732236, 0.05045047, 0.14597263 }, { 0.0788708,
0.05305461, 0.14995981 }, { 0.08450105, 0.05559631, 0.15396203 }, { 0.09011319, 0.05808059, 0.15797687
}, { 0.09572396, 0.06050127, 0.16200507 }, { 0.10132312, 0.06286782, 0.16604287 }, { 0.10692823,
0.06517224, 0.17009175 }, { 0.1125315, 0.06742194, 0.17414848 }, { 0.11813947, 0.06961499, 0.17821272
}, { 0.12375803, 0.07174938, 0.18228425 }, { 0.12938228, 0.07383015, 0.18636053 }, { 0.13501631,
0.07585609, 0.19044109 }, { 0.14066867, 0.0778224, 0.19452676 }, { 0.14633406, 0.07973393, 0.1986151
}, { 0.15201338, 0.08159108, 0.20270523 }, { 0.15770877, 0.08339312, 0.20679668 }, { 0.16342174,
0.0851396, 0.21088893 }, { 0.16915387, 0.08682996, 0.21498104 }, { 0.17489524, 0.08848235, 0.2190294
}, { 0.18065495, 0.09009031, 0.22303512 }, { 0.18643324, 0.09165431, 0.22699705 }, { 0.19223028,
0.09317479, 0.23091409 }, { 0.19804623, 0.09465217, 0.23478512 }, { 0.20388117, 0.09608689, 0.23860907
}, { 0.20973515, 0.09747934, 0.24238489 }, { 0.21560818, 0.09882993, 0.24611154 }, { 0.22150014,
0.10013944, 0.2497868 }, { 0.22741085, 0.10140876, 0.25340813 }, { 0.23334047, 0.10263737, 0.25697736
}, { 0.23928891, 0.10382562, 0.2604936 }, { 0.24525608, 0.10497384, 0.26395596 }, { 0.25124182,
0.10608236, 0.26736359 }, { 0.25724602, 0.10715148, 0.27071569 }, { 0.26326851, 0.1081815, 0.27401148
}, { 0.26930915, 0.1091727, 0.2772502 }, { 0.27536766, 0.11012568, 0.28043021 }, { 0.28144375,
0.11104133, 0.2835489 }, { 0.2875374, 0.11191896, 0.28660853 }, { 0.29364846, 0.11275876, 0.2896085 },
{ 0.29977678, 0.11356089, 0.29254823 }, { 0.30592213, 0.11432553, 0.29542718 }, { 0.31208435,
0.11505284, 0.29824485 }, { 0.31826327, 0.1157429, 0.30100076 }, { 0.32445869, 0.11639585, 0.30369448
}, { 0.33067031, 0.11701189, 0.30632563 }, { 0.33689808, 0.11759095, 0.3088938 }, { 0.34314168,
0.11813362, 0.31139721 }, { 0.34940101, 0.11863987, 0.3138355 }, { 0.355676, 0.11910909, 0.31620996 },
{ 0.36196644, 0.1195413, 0.31852037 }, { 0.36827206, 0.11993653, 0.32076656 }, { 0.37459292,
0.12029443, 0.32294825 }, { 0.38092887, 0.12061482, 0.32506528 }, { 0.38727975, 0.12089756, 0.3271175
}, { 0.39364518, 0.12114272, 0.32910494 }, { 0.40002537, 0.12134964, 0.33102734 }, { 0.40642019,
0.12151801, 0.33288464 }, { 0.41282936, 0.12164769, 0.33467689 }, { 0.41925278, 0.12173833, 0.33640407
}, { 0.42569057, 0.12178916, 0.33806605 }, { 0.43214263, 0.12179973, 0.33966284 }, { 0.43860848,
0.12177004, 0.34119475 }, { 0.44508855, 0.12169883, 0.34266151 }, { 0.45158266, 0.12158557, 0.34406324
}, { 0.45809049, 0.12142996, 0.34540024 }, { 0.46461238, 0.12123063, 0.34667231 }, { 0.47114798,
0.12098721, 0.34787978 }, { 0.47769736, 0.12069864, 0.34902273 }, { 0.48426077, 0.12036349, 0.35010104
}, { 0.49083761, 0.11998161, 0.35111537 }, { 0.49742847, 0.11955087, 0.35206533 }, { 0.50403286,
0.11907081, 0.35295152 }, { 0.51065109, 0.11853959, 0.35377385 }, { 0.51728314, 0.1179558, 0.35453252
}, { 0.52392883, 0.11731817, 0.35522789 }, { 0.53058853, 0.11662445, 0.35585982 }, { 0.53726173,
0.11587369, 0.35642903 }, { 0.54394898, 0.11506307, 0.35693521 }, { 0.5506426, 0.11420757, 0.35737863
}, { 0.55734473, 0.11330456, 0.35775059 }, { 0.56405586, 0.11235265, 0.35804813 }, { 0.57077365,
0.11135597, 0.35827146 }, { 0.5774991, 0.11031233, 0.35841679 }, { 0.58422945, 0.10922707, 0.35848469
}, { 0.59096382, 0.10810205, 0.35847347 }, { 0.59770215, 0.10693774, 0.35838029 }, { 0.60444226,
0.10573912, 0.35820487 }, { 0.61118304, 0.10450943, 0.35794557 }, { 0.61792306, 0.10325288, 0.35760108
}, { 0.62466162, 0.10197244, 0.35716891 }, { 0.63139686, 0.10067417, 0.35664819 }, { 0.63812122,
0.09938212, 0.35603757 }, { 0.64483795, 0.0980891, 0.35533555 }, { 0.65154562, 0.09680192, 0.35454107
}, { 0.65824241, 0.09552918, 0.3536529 }, { 0.66492652, 0.09428017, 0.3526697 }, { 0.67159578,
0.09306598, 0.35159077 }, { 0.67824099, 0.09192342, 0.3504148 }, { 0.684863, 0.09085633, 0.34914061 },
{ 0.69146268, 0.0898675, 0.34776864 }, { 0.69803757, 0.08897226, 0.3462986 }, { 0.70457834, 0.0882129,
0.34473046 }, { 0.71108138, 0.08761223, 0.3430635 }, { 0.7175507, 0.08716212, 0.34129974 }, {
0.72398193, 0.08688725, 0.33943958 }, { 0.73035829, 0.0868623, 0.33748452 }, { 0.73669146, 0.08704683,
0.33543669 }, { 0.74297501, 0.08747196, 0.33329799 }, { 0.74919318, 0.08820542, 0.33107204 }, {
0.75535825, 0.08919792, 0.32876184 }, { 0.76145589, 0.09050716, 0.32637117 }, { 0.76748424,
0.09213602, 0.32390525 }, { 0.77344838, 0.09405684, 0.32136808 }, { 0.77932641, 0.09634794, 0.31876642
}, { 0.78513609, 0.09892473, 0.31610488 }, { 0.79085854, 0.10184672, 0.313391 }, { 0.7965014,
0.10506637, 0.31063031 }, { 0.80205987, 0.10858333, 0.30783 }, { 0.80752799, 0.11239964, 0.30499738 },
{ 0.81291606, 0.11645784, 0.30213802 }, { 0.81820481, 0.12080606, 0.29926105 }, { 0.82341472,
0.12535343, 0.2963705 }, { 0.82852822, 0.13014118, 0.29347474 }, { 0.83355779, 0.13511035, 0.29057852
}, { 0.83850183, 0.14025098, 0.2876878 }, { 0.84335441, 0.14556683, 0.28480819 }, { 0.84813096,
0.15099892, 0.281943 }, { 0.85281737, 0.15657772, 0.27909826 }, { 0.85742602, 0.1622583, 0.27627462 },
{ 0.86196552, 0.16801239, 0.27346473 }, { 0.86641628, 0.17387796, 0.27070818 }, { 0.87079129,
0.17982114, 0.26797378 }, { 0.87507281, 0.18587368, 0.26529697 }, { 0.87925878, 0.19203259, 0.26268136
}, { 0.8833417, 0.19830556, 0.26014181 }, { 0.88731387, 0.20469941, 0.25769539 }, { 0.89116859,
0.21121788, 0.2553592 }, { 0.89490337, 0.21785614, 0.25314362 }, { 0.8985026, 0.22463251, 0.25108745
}, { 0.90197527, 0.23152063, 0.24918223 }, { 0.90530097, 0.23854541, 0.24748098 }, { 0.90848638,
0.24568473, 0.24598324 }, { 0.911533, 0.25292623, 0.24470258 }, { 0.9144225, 0.26028902, 0.24369359 },
{ 0.91717106, 0.26773821, 0.24294137 }, { 0.91978131, 0.27526191, 0.24245973 }, { 0.92223947,
0.28287251, 0.24229568 }, { 0.92456587, 0.29053388, 0.24242622 }, { 0.92676657, 0.29823282, 0.24285536
}, { 0.92882964, 0.30598085, 0.24362274 }, { 0.93078135, 0.31373977, 0.24468803 }, { 0.93262051,
0.3215093, 0.24606461 }, { 0.93435067, 0.32928362, 0.24775328 }, { 0.93599076, 0.33703942, 0.24972157
}, { 0.93752831, 0.34479177, 0.25199928 }, { 0.93899289, 0.35250734, 0.25452808 }, { 0.94036561,
0.36020899, 0.25734661 }, { 0.94167588, 0.36786594, 0.2603949 }, { 0.94291042, 0.37549479, 0.26369821
}, { 0.94408513, 0.3830811, 0.26722004 }, { 0.94520419, 0.39062329, 0.27094924 }, { 0.94625977,
0.39813168, 0.27489742 }, { 0.94727016, 0.4055909, 0.27902322 }, { 0.94823505, 0.41300424, 0.28332283
}, { 0.94914549, 0.42038251, 0.28780969 }, { 0.95001704, 0.42771398, 0.29244728 }, { 0.95085121,
0.43500005, 0.29722817 }, { 0.95165009, 0.44224144, 0.30214494 }, { 0.9524044, 0.44944853, 0.3072105
}, { 0.95312556, 0.45661389, 0.31239776 }, { 0.95381595, 0.46373781, 0.31769923 }, { 0.95447591,
0.47082238, 0.32310953 }, { 0.95510255, 0.47787236, 0.32862553 }, { 0.95569679, 0.48489115, 0.33421404
}, { 0.95626788, 0.49187351, 0.33985601 }, { 0.95681685, 0.49882008, 0.34555431 }, { 0.9573439,
0.50573243, 0.35130912 }, { 0.95784842, 0.51261283, 0.35711942 }, { 0.95833051, 0.51946267, 0.36298589
```

```
      }, { 0.95879054, 0.52628305, 0.36890904 }, { 0.95922872, 0.53307513, 0.3748895 }, { 0.95964538,
      0.53983991, 0.38092784 }, { 0.96004345, 0.54657593, 0.3870292 }, { 0.96042097, 0.55328624, 0.39319057
      }, { 0.96077819, 0.55997184, 0.39941173 }, { 0.9611152, 0.5666337, 0.40569343 }, { 0.96143273,
      0.57327231, 0.41203603 }, { 0.96173392, 0.57988594, 0.41844491 }, { 0.96201757, 0.58647675, 0.42491751
      }, { 0.96228344, 0.59304598, 0.43145271 }, { 0.96253168, 0.5995944, 0.43805131 }, { 0.96276513,
      0.60612062, 0.44471698 }, { 0.96298491, 0.6126247, 0.45145074 }, { 0.96318967, 0.61910879, 0.45824902
      }, { 0.96337949, 0.6255736, 0.46511271 }, { 0.96355923, 0.63201624, 0.47204746 }, { 0.96372785,
      0.63843852, 0.47905028 }, { 0.96388426, 0.64484214, 0.4861196 }, { 0.96403203, 0.65122535, 0.4932578
      }, { 0.96417332, 0.65758729, 0.50046894 }, { 0.9643063, 0.66393045, 0.5077467 }, { 0.96443322,
      0.67025402, 0.51509334 }, { 0.96455845, 0.67655564, 0.52251447 }, { 0.96467922, 0.68283846, 0.53000231
      }, { 0.96479861, 0.68910113, 0.53756026 }, { 0.96492035, 0.69534192, 0.5451917 }, { 0.96504223,
      0.7015636, 0.5528892 }, { 0.96516917, 0.70776351, 0.5606593 }, { 0.96530224, 0.71394212, 0.56849894 },
      { 0.96544032, 0.72010124, 0.57640375 }, { 0.96559206, 0.72623592, 0.58438387 }, { 0.96575293,
      0.73235058, 0.59242739 }, { 0.96592829, 0.73844258, 0.60053991 }, { 0.96612013, 0.74451182, 0.60871954
      }, { 0.96632832, 0.75055966, 0.61696136 }, { 0.96656022, 0.75658231, 0.62527295 }, { 0.96681185,
      0.76258381, 0.63364277 }, { 0.96709183, 0.76855969, 0.64207921 }, { 0.96739773, 0.77451297, 0.65057302
      }, { 0.96773482, 0.78044149, 0.65912731 }, { 0.96810471, 0.78634563, 0.66773889 }, { 0.96850919,
      0.79222565, 0.6764046 }, { 0.96893132, 0.79809112, 0.68512266 }, { 0.96935926, 0.80395415, 0.69383201
      }, { 0.9698028, 0.80981139, 0.70252255 }, { 0.97025511, 0.81566605, 0.71120296 }, { 0.97071849,
      0.82151775, 0.71987163 }, { 0.97120159, 0.82736371, 0.72851999 }, { 0.97169389, 0.83320847, 0.73716071
      }, { 0.97220061, 0.83905052, 0.74578903 }, { 0.97272597, 0.84488881, 0.75440141 }, { 0.97327085,
      0.85072354, 0.76299805 }, { 0.97383206, 0.85655639, 0.77158353 }, { 0.97441222, 0.86238689, 0.78015619
      }, { 0.97501782, 0.86821321, 0.78871034 }, { 0.97564391, 0.87403763, 0.79725261 }, { 0.97628674,
      0.87986189, 0.8057883 }, { 0.97696114, 0.88568129, 0.81430324 }, { 0.97765722, 0.89149971, 0.82280948
      }, { 0.97837585, 0.89731727, 0.83130786 }, { 0.97912374, 0.90313207, 0.83979337 }, { 0.979891,
      0.90894778, 0.84827858 }, { 0.98067764, 0.91476465, 0.85676611 }, { 0.98137749, 0.92061729, 0.86536915
      } };
00066         private static double[,] MakoRGB = new double[256, 3] { { 0.04503935, 0.01482344, 0.02092227
      }, { 0.04933018, 0.01709292, 0.02535719 }, { 0.05356262, 0.01950702, 0.03018802 }, { 0.05774337,
      0.02205989, 0.03545515 }, { 0.06188095, 0.02474764, 0.04115287 }, { 0.06598247, 0.0275665, 0.04691409
      }, { 0.07005374, 0.03051278, 0.05264306 }, { 0.07409947, 0.03358324, 0.05834631 }, { 0.07812339,
      0.03677446, 0.06403249 }, { 0.08212852, 0.0400833, 0.06970862 }, { 0.08611731, 0.04339148, 0.07538208
      }, { 0.09009161, 0.04664706, 0.08105568 }, { 0.09405308, 0.04985685, 0.08673591 }, { 0.09800301,
      0.05302279, 0.09242646 }, { 0.10194255, 0.05614641, 0.09813162 }, { 0.10587261, 0.05922941, 0.103854
      }, { 0.1097942, 0.06227277, 0.10959847 }, { 0.11370826, 0.06527747, 0.11536893 }, { 0.11761516,
      0.06824548, 0.12116393 }, { 0.12151575, 0.07117741, 0.12698763 }, { 0.12541095, 0.07407363, 0.1328442
      }, { 0.12930083, 0.07693611, 0.13873064 }, { 0.13317849, 0.07976988, 0.14465095 }, { 0.13701138,
      0.08259683, 0.15060265 }, { 0.14079223, 0.08542126, 0.15659379 }, { 0.14452486, 0.08824175, 0.16262484
      }, { 0.14820351, 0.09106304, 0.16869476 }, { 0.15183185, 0.09388372, 0.17480366 }, { 0.15540398,
      0.09670855, 0.18094993 }, { 0.15892417, 0.09953561, 0.18713384 }, { 0.16238588, 0.10236998, 0.19335329
      }, { 0.16579435, 0.10520905, 0.19960847 }, { 0.16914226, 0.10805832, 0.20589698 }, { 0.17243586,
      0.11091443, 0.21221911 }, { 0.17566717, 0.11378321, 0.21857219 }, { 0.17884322, 0.11666074, 0.2249565
      }, { 0.18195582, 0.11955283, 0.23136943 }, { 0.18501213, 0.12245547, 0.23781116 }, { 0.18800459,
      0.12537395, 0.24427914 }, { 0.19093944, 0.1283047, 0.25077369 }, { 0.19381092, 0.13125179, 0.25729255
      }, { 0.19662307, 0.13421303, 0.26383543 }, { 0.19937337, 0.13719028, 0.27040111 }, { 0.20206187,
      0.14018372, 0.27698891 }, { 0.20469116, 0.14319196, 0.28359861 }, { 0.20725547, 0.14621882, 0.29022775
      }, { 0.20976258, 0.14925954, 0.29687795 }, { 0.21220409, 0.15231929, 0.30354703 }, { 0.21458611,
      0.15539445, 0.31023563 }, { 0.21690827, 0.15848519, 0.31694355 }, { 0.21916481, 0.16159489, 0.32366939
      }, { 0.2213631, 0.16471913, 0.33041431 }, { 0.22349947, 0.1678599, 0.33717781 }, { 0.2255714,
      0.1710185, 0.34395925 }, { 0.22758415, 0.17419169, 0.35075983 }, { 0.22953569, 0.17738041, 0.35757941
      }, { 0.23142077, 0.18058733, 0.3644173 }, { 0.2332454, 0.18380872, 0.37127514 }, { 0.2350092,
      0.18704459, 0.3781528 }, { 0.23670785, 0.190297, 0.38504973 }, { 0.23834119, 0.19356547, 0.39196711 },
      { 0.23991189, 0.19684817, 0.39890581 }, { 0.24141903, 0.20014508, 0.4058667 }, { 0.24286214,
      0.20345642, 0.4128484 }, { 0.24423453, 0.20678459, 0.41985299 }, { 0.24554109, 0.21012669, 0.42688124
      }, { 0.2467815, 0.21348266, 0.43393244 }, { 0.24795393, 0.21685249, 0.4410088 }, { 0.24905614,
      0.22023618, 0.448113 }, { 0.25007383, 0.22365053, 0.45519562 }, { 0.25098926, 0.22710664, 0.46223892
      }, { 0.25179696, 0.23060342, 0.46925447 }, { 0.25249346, 0.23414353, 0.47623196 }, { 0.25307401,
      0.23772973, 0.48316271 }, { 0.25353152, 0.24136961, 0.49001976 }, { 0.25386167, 0.24506548, 0.49679407
      }, { 0.25406082, 0.2488164, 0.50348932 }, { 0.25412435, 0.25262843, 0.51007843 }, { 0.25404842,
      0.25650743, 0.51653282 }, { 0.25383134, 0.26044852, 0.52286845 }, { 0.2534705, 0.26446165, 0.52903422
      }, { 0.25296722, 0.2685428, 0.53503572 }, { 0.2523226, 0.27269346, 0.54085315 }, { 0.25153974,
      0.27691629, 0.54645752 }, { 0.25062402, 0.28120467, 0.55185939 }, { 0.24958205, 0.28556371, 0.55701246
      }, { 0.24842386, 0.28998148, 0.56194601 }, { 0.24715928, 0.29446327, 0.56660884 }, { 0.24580099,
      0.29899398, 0.57104399 }, { 0.24436202, 0.30357852, 0.57519929 }, { 0.24285591, 0.30819938, 0.57913247
      }, { 0.24129828, 0.31286235, 0.58278615 }, { 0.23970131, 0.3175495, 0.5862272 }, { 0.23807973,
      0.32226344, 0.58941872 }, { 0.23644557, 0.32699241, 0.59240198 }, { 0.2348113, 0.33173196, 0.59518282
      }, { 0.23318874, 0.33648036, 0.59775543 }, { 0.2315855, 0.34122763, 0.60016456 }, { 0.23001121,
      0.34597357, 0.60240251 }, { 0.2284748, 0.35071512, 0.6044784 }, { 0.22698081, 0.35544612, 0.60642528
      }, { 0.22553305, 0.36016515, 0.60825252 }, { 0.22413977, 0.36487341, 0.60994938 }, { 0.22280246,
      0.36956728, 0.61154118 }, { 0.22152555, 0.37424409, 0.61304472 }, { 0.22030752, 0.37890437, 0.61446646
      }, { 0.2191538, 0.38354668, 0.61581561 }, { 0.21806257, 0.38817169, 0.61709794 }, { 0.21703799,
      0.39277882, 0.61831922 }, { 0.21607792, 0.39736958, 0.61948028 }, { 0.21518463, 0.40194196, 0.62059763
      }, { 0.21435467, 0.40649717, 0.62167507 }, { 0.21358663, 0.41103579, 0.62271724 }, { 0.21288172,
      0.41555771, 0.62373011 }, { 0.21223835, 0.42006355, 0.62471794 }, { 0.21165312, 0.42455441, 0.62568371
      }, { 0.21112526, 0.42903064, 0.6266318 }, { 0.21065161, 0.43349321, 0.62756504 }, { 0.21023306,
      0.43794288, 0.62848279 }, { 0.20985996, 0.44238227, 0.62938329 }, { 0.20951045, 0.44680966, 0.63030696
      }, { 0.20916709, 0.45122981, 0.63124483 }, { 0.20882976, 0.45564335, 0.63219599 }, { 0.20849798,
      0.46005094, 0.63315928 }, { 0.20817199, 0.46445309, 0.63413391 }, { 0.20785149, 0.46885041, 0.63511876
      }, { 0.20753716, 0.47324327, 0.63611321 }, { 0.20722876, 0.47763224, 0.63711608 }, { 0.20692679,
      0.48201774, 0.63812656 }, { 0.20663156, 0.48640018, 0.63914367 }, { 0.20634336, 0.49078002, 0.64016638
      }, { 0.20606303, 0.49515755, 0.6411939 }, { 0.20578999, 0.49953341, 0.64222457 }, { 0.20552612,
      0.50390766, 0.64325811 }, { 0.20527189, 0.50828072, 0.64429331 }, { 0.20502868, 0.51265277, 0.64532947
      }, { 0.20479718, 0.51702417, 0.64636539 }, { 0.20457804, 0.52139527, 0.64739979 }, { 0.20437304,
      0.52576622, 0.64843198 }, { 0.20418396, 0.53013715, 0.64946117 }, { 0.20401238, 0.53450825, 0.65048638
      }, { 0.20385896, 0.53887991, 0.65150606 }, { 0.20372653, 0.54325208, 0.65251978 }, { 0.20361709,
      0.5476249, 0.6535266 }, { 0.20353258, 0.55199854, 0.65452542 }, { 0.20347472, 0.55637318, 0.655515 },
      { 0.20344718, 0.56074869, 0.65649508 }, { 0.20345161, 0.56512531, 0.65746419 }, { 0.20349089,
```

```
0.56950304, 0.65842151 }, { 0.20356842, 0.57388184, 0.65936642 }, { 0.20368663, 0.57826181, 0.66029768
}, { 0.20384884, 0.58264293, 0.6612145 }, { 0.20405904, 0.58702506, 0.66211645 }, { 0.20431921,
0.59140842, 0.66300179 }, { 0.20463464, 0.59579264, 0.66387079 }, { 0.20500731, 0.60017798, 0.66472159
}, { 0.20544449, 0.60456387, 0.66555409 }, { 0.20596097, 0.60894927, 0.66636568 }, { 0.20654832,
0.61333521, 0.66715744 }, { 0.20721003, 0.61772167, 0.66792838 }, { 0.20795035, 0.62210845, 0.66867802
}, { 0.20877302, 0.62649546, 0.66940555 }, { 0.20968223, 0.63088252, 0.6701105 }, { 0.21068163,
0.63526951, 0.67079211 }, { 0.21177544, 0.63965621, 0.67145005 }, { 0.21298582, 0.64404072, 0.67208182
}, { 0.21430361, 0.64842404, 0.67268861 }, { 0.21572716, 0.65280655, 0.67326978 }, { 0.21726052,
0.65718791, 0.6738255 }, { 0.21890636, 0.66156803, 0.67435491 }, { 0.220668, 0.66594665, 0.67485792 },
{ 0.22255447, 0.67032297, 0.67533374 }, { 0.22458372, 0.67469531, 0.67578061 }, { 0.22673713,
0.67906542, 0.67620044 }, { 0.22901625, 0.6834332, 0.67659251 }, { 0.23142316, 0.68779836, 0.67695703
}, { 0.23395924, 0.69216072, 0.67729378 }, { 0.23663857, 0.69651881, 0.67760151 }, { 0.23946645,
0.70087194, 0.67788018 }, { 0.24242624, 0.70522162, 0.67813088 }, { 0.24549008, 0.70957083, 0.67835215
}, { 0.24863372, 0.71392166, 0.67854868 }, { 0.25187832, 0.71827158, 0.67872193 }, { 0.25524083,
0.72261873, 0.67887024 }, { 0.25870947, 0.72696469, 0.67898912 }, { 0.26229238, 0.73130855, 0.67907645
}, { 0.26604085, 0.73564353, 0.67914062 }, { 0.26993099, 0.73997282, 0.67917264 }, { 0.27397488,
0.74429484, 0.67917096 }, { 0.27822463, 0.74860229, 0.67914468 }, { 0.28264201, 0.75290034, 0.67907959
}, { 0.2873016, 0.75717817, 0.67899164 }, { 0.29215894, 0.76144162, 0.67886578 }, { 0.29729823,
0.76567816, 0.67871894 }, { 0.30268199, 0.76989232, 0.67853896 }, { 0.30835665, 0.77407636, 0.67833512
}, { 0.31435139, 0.77822478, 0.67811118 }, { 0.3206671, 0.78233575, 0.67786729 }, { 0.32733158,
0.78640315, 0.67761027 }, { 0.33437168, 0.79042043, 0.67734882 }, { 0.34182112, 0.79437948, 0.67709394
}, { 0.34968889, 0.79827511, 0.67685638 }, { 0.35799244, 0.80210037, 0.67664969 }, { 0.36675371,
0.80584651, 0.67649539 }, { 0.3759816, 0.80950627, 0.67641393 }, { 0.38566792, 0.81307432, 0.67642947
}, { 0.39579804, 0.81654592, 0.67656899 }, { 0.40634556, 0.81991799, 0.67686215 }, { 0.41730243,
0.82318339, 0.67735255 }, { 0.4285828, 0.82635051, 0.6780564 }, { 0.44012728, 0.82942353, 0.67900049
}, { 0.45189421, 0.83240398, 0.68021733 }, { 0.46378379, 0.83530763, 0.6817062 }, { 0.47573199,
0.83814472, 0.68347352 }, { 0.48769865, 0.84092197, 0.68552698 }, { 0.49962354, 0.84365379, 0.68783929
}, { 0.5114027, 0.8463718, 0.69029789 }, { 0.52301693, 0.84908401, 0.69288545 }, { 0.53447549,
0.85179048, 0.69561066 }, { 0.54578602, 0.8544913, 0.69848331 }, { 0.55695565, 0.85718723, 0.70150427
}, { 0.56798832, 0.85987893, 0.70468261 }, { 0.57888639, 0.86256715, 0.70802931 }, { 0.5896541,
0.8652532, 0.71154204 }, { 0.60028928, 0.86793835, 0.71523675 }, { 0.61079441, 0.87062438, 0.71910895
}, { 0.62116633, 0.87331311, 0.72317003 }, { 0.63140509, 0.87600675, 0.72741689 }, { 0.64150735,
0.87870746, 0.73185717 }, { 0.65147219, 0.8814179, 0.73648495 }, { 0.66129632, 0.8841403, 0.74130658
}, { 0.67097934, 0.88687758, 0.74631123 }, { 0.68051833, 0.88963189, 0.75150483 }, { 0.68991419,
0.89240612, 0.75687187 }, { 0.69916533, 0.89520211, 0.76241714 }, { 0.70827373, 0.89802257, 0.76812286
}, { 0.71723995, 0.90086891, 0.77399039 }, { 0.72606665, 0.90374337, 0.7800041 }, { 0.73475675,
0.90664718, 0.78615802 }, { 0.74331358, 0.90958151, 0.79244474 }, { 0.75174143, 0.91254787, 0.79884925
}, { 0.76004473, 0.91554656, 0.80536823 }, { 0.76827704, 0.91856549, 0.81196513 }, { 0.77647029,
0.921603, 0.81855729 }, { 0.78462009, 0.92466151, 0.82514119 }, { 0.79273542, 0.92773848, 0.83172131
}, { 0.8008109, 0.93083672, 0.83829355 }, { 0.80885107, 0.93395528, 0.84485982 }, { 0.81685878,
0.9370938, 0.85142101 }, { 0.82483206, 0.94025378, 0.8579751 }, { 0.83277661, 0.94343371, 0.86452477
}, { 0.84069127, 0.94663473, 0.87106853 }, { 0.84857662, 0.9498573, 0.8776059 }, { 0.8564431,
0.95309792, 0.88414253 }, { 0.86429066, 0.95635719, 0.89067759 }, { 0.87218969, 0.95960708, 0.89725384
} };
00067         private static double[,] TurboRGB = new double[256, 3] { { 0.18995, 0.07176, 0.23217 }, {
0.19483, 0.08339, 0.26149 }, { 0.19956, 0.09498, 0.29024 }, { 0.20415, 0.10652, 0.31844 }, { 0.2086,
0.11802, 0.34607 }, { 0.21291, 0.12947, 0.37314 }, { 0.21708, 0.14087, 0.39964 }, { 0.22111, 0.15223,
0.42558 }, { 0.225, 0.16354, 0.45096 }, { 0.22875, 0.17481, 0.47578 }, { 0.23236, 0.18603, 0.50004 },
{ 0.23582, 0.1972, 0.52373 }, { 0.23915, 0.20833, 0.54686 }, { 0.24234, 0.21941, 0.56942 }, { 0.24539,
0.23044, 0.59142 }, { 0.2483, 0.24143, 0.61286 }, { 0.25107, 0.25237, 0.63374 }, { 0.25369, 0.26327,
0.65406 }, { 0.25618, 0.27412, 0.67381 }, { 0.25853, 0.28492, 0.693 }, { 0.26074, 0.29568, 0.71162 },
{ 0.2628, 0.30639, 0.72968 }, { 0.26473, 0.31706, 0.74718 }, { 0.26652, 0.32768, 0.76412 }, { 0.26816,
0.33825, 0.7805 }, { 0.26967, 0.34878, 0.79631 }, { 0.27103, 0.35926, 0.81156 }, { 0.27226, 0.3697,
0.82624 }, { 0.27334, 0.38008, 0.84037 }, { 0.27429, 0.39043, 0.85393 }, { 0.27509, 0.40072, 0.86692
}, { 0.27576, 0.41097, 0.87936 }, { 0.27628, 0.42118, 0.89123 }, { 0.27667, 0.43134, 0.90254 }, {
0.27691, 0.44145, 0.91328 }, { 0.27701, 0.45152, 0.92347 }, { 0.27698, 0.46153, 0.93309 }, { 0.2768,
0.47151, 0.94214 }, { 0.27648, 0.48144, 0.95064 }, { 0.27603, 0.49132, 0.95857 }, { 0.27543, 0.50115,
0.96594 }, { 0.27469, 0.51094, 0.97275 }, { 0.27381, 0.52069, 0.97899 }, { 0.27273, 0.5304, 0.98461 },
{ 0.27106, 0.54015, 0.9893 }, { 0.26878, 0.54995, 0.99303 }, { 0.26592, 0.55979, 0.99583 }, { 0.26252,
0.56967, 0.99773 }, { 0.25862, 0.57958, 0.99876 }, { 0.25425, 0.5895, 0.99896 }, { 0.24946, 0.59943,
0.99835 }, { 0.24427, 0.60937, 0.99697 }, { 0.23874, 0.61931, 0.99485 }, { 0.23288, 0.62923, 0.99202
}, { 0.22676, 0.63913, 0.98851 }, { 0.22039, 0.64901, 0.98436 }, { 0.21382, 0.65886, 0.97959 }, {
0.20708, 0.66866, 0.97423 }, { 0.20021, 0.67842, 0.96833 }, { 0.19326, 0.68812, 0.9619 }, { 0.18625,
0.69775, 0.95498 }, { 0.17923, 0.70732, 0.94761 }, { 0.17223, 0.7168, 0.93981 }, { 0.16529, 0.7262,
0.93161 }, { 0.15844, 0.73551, 0.92305 }, { 0.15173, 0.74472, 0.91416 }, { 0.14519, 0.75381, 0.90496
}, { 0.13886, 0.76279, 0.8955 }, { 0.13278, 0.77165, 0.8858 }, { 0.12698, 0.78037, 0.8759 }, {
0.12151, 0.78896, 0.86581 }, { 0.11639, 0.7974, 0.85559 }, { 0.11167, 0.80569, 0.84525 }, { 0.10738,
0.81381, 0.83484 }, { 0.10357, 0.82177, 0.82437 }, { 0.10026, 0.82955, 0.81389 }, { 0.0975, 0.83714,
0.80342 }, { 0.09532, 0.84455, 0.79299 }, { 0.09377, 0.85175, 0.78264 }, { 0.09287, 0.85875, 0.7724 },
{ 0.09267, 0.86554, 0.7623 }, { 0.0932, 0.87211, 0.75237 }, { 0.09451, 0.87844, 0.74265 }, { 0.09662,
0.88454, 0.73316 }, { 0.09958, 0.8904, 0.72393 }, { 0.10342, 0.896, 0.715 }, { 0.10815, 0.90142,
0.70599 }, { 0.11374, 0.90673, 0.69651 }, { 0.12014, 0.91193, 0.6866 }, { 0.12733, 0.91701, 0.67627 },
{ 0.13526, 0.92197, 0.66556 }, { 0.14391, 0.9268, 0.65448 }, { 0.15323, 0.93151, 0.64308 }, { 0.16319,
0.93609, 0.63137 }, { 0.17377, 0.94053, 0.61938 }, { 0.18491, 0.94484, 0.60713 }, { 0.19659, 0.94901,
0.59466 }, { 0.20877, 0.95304, 0.58199 }, { 0.22142, 0.95692, 0.56914 }, { 0.23449, 0.96065, 0.55614
}, { 0.24797, 0.96423, 0.54303 }, { 0.2618, 0.96765, 0.52981 }, { 0.27597, 0.97092, 0.51653 }, {
0.29042, 0.97403, 0.50321 }, { 0.30513, 0.97697, 0.48987 }, { 0.32006, 0.97974, 0.47654 }, { 0.33517,
0.98234, 0.46325 }, { 0.35043, 0.98477, 0.45002 }, { 0.36581, 0.98702, 0.43688 }, { 0.38127, 0.98909,
0.42386 }, { 0.39678, 0.99098, 0.41098 }, { 0.41229, 0.99268, 0.39826 }, { 0.42778, 0.99419, 0.38575
}, { 0.44321, 0.99551, 0.37345 }, { 0.45854, 0.99663, 0.3614 }, { 0.47375, 0.99755, 0.34963 }, {
0.48879, 0.99828, 0.33816 }, { 0.50362, 0.99879, 0.32701 }, { 0.51822, 0.9991, 0.31622 }, { 0.53255,
0.99919, 0.30581 }, { 0.54658, 0.99907, 0.29581 }, { 0.56026, 0.99873, 0.28623 }, { 0.57357, 0.99817,
0.27712 }, { 0.58646, 0.99739, 0.26849 }, { 0.59891, 0.99638, 0.26038 }, { 0.61088, 0.99514, 0.2528 },
{ 0.62233, 0.99366, 0.24579 }, { 0.63323, 0.99195, 0.23937 }, { 0.64362, 0.98999, 0.23356 }, {
0.65394, 0.98775, 0.22835 }, { 0.66428, 0.98524, 0.2237 }, { 0.67462, 0.98246, 0.2196 }, { 0.68494,
0.97941, 0.21602 }, { 0.69525, 0.9761, 0.21294 }, { 0.70553, 0.97255, 0.21032 }, { 0.71577, 0.96875,
0.20815 }, { 0.72596, 0.9647, 0.2064 }, { 0.7361, 0.96043, 0.20504 }, { 0.74617, 0.95593, 0.20406 }, {
```

```
      0.75617, 0.95121, 0.20343 }, { 0.76608, 0.94627, 0.20311 }, { 0.77591, 0.94113, 0.2031 }, { 0.78563,
      0.93579, 0.20336 }, { 0.79524, 0.93025, 0.20386 }, { 0.80473, 0.92452, 0.20459 }, { 0.8141, 0.91861,
      0.20552 }, { 0.82333, 0.91253, 0.20663 }, { 0.83241, 0.90627, 0.20788 }, { 0.84133, 0.89986, 0.20926
      }, { 0.8501, 0.89328, 0.21074 }, { 0.85868, 0.88655, 0.2123 }, { 0.86709, 0.87968, 0.21391 }, {
      0.8753, 0.87267, 0.21555 }, { 0.88331, 0.86553, 0.21719 }, { 0.89112, 0.85826, 0.2188 }, { 0.8987,
      0.85087, 0.22038 }, { 0.90605, 0.84337, 0.22188 }, { 0.91317, 0.83576, 0.22328 }, { 0.92004, 0.82806,
      0.22456 }, { 0.92666, 0.82025, 0.2257 }, { 0.93301, 0.81236, 0.22667 }, { 0.93909, 0.80439, 0.22744 },
      { 0.94489, 0.79634, 0.228 }, { 0.95039, 0.78823, 0.22831 }, { 0.9556, 0.78005, 0.22836 }, { 0.96049,
      0.77181, 0.22811 }, { 0.96507, 0.76352, 0.22754 }, { 0.96931, 0.75519, 0.22663 }, { 0.97323, 0.74682,
      0.22536 }, { 0.97679, 0.73842, 0.22369 }, { 0.98, 0.73, 0.22161 }, { 0.98289, 0.7214, 0.21918 }, {
      0.98549, 0.7125, 0.2165 }, { 0.98781, 0.7033, 0.21358 }, { 0.98986, 0.69382, 0.21043 }, { 0.99163,
      0.68408, 0.20706 }, { 0.99314, 0.67408, 0.20348 }, { 0.99438, 0.66386, 0.19971 }, { 0.99535, 0.65341,
      0.19577 }, { 0.99607, 0.64277, 0.19165 }, { 0.99654, 0.63193, 0.18738 }, { 0.99675, 0.62093, 0.18297
      }, { 0.99672, 0.60977, 0.17842 }, { 0.99644, 0.59846, 0.17376 }, { 0.99593, 0.58703, 0.16899 }, {
      0.99517, 0.57549, 0.16412 }, { 0.99419, 0.56386, 0.15918 }, { 0.99297, 0.55214, 0.15417 }, { 0.99153,
      0.54036, 0.1491 }, { 0.98987, 0.52854, 0.14398 }, { 0.98799, 0.51667, 0.13883 }, { 0.9859, 0.50479,
      0.13367 }, { 0.9836, 0.49291, 0.12849 }, { 0.98108, 0.48104, 0.12332 }, { 0.97837, 0.4692, 0.11817 },
      { 0.97545, 0.4574, 0.11305 }, { 0.97234, 0.44565, 0.10797 }, { 0.96904, 0.43399, 0.10294 }, { 0.96555,
      0.42241, 0.09798 }, { 0.96187, 0.41093, 0.0931 }, { 0.95801, 0.39958, 0.08831 }, { 0.95398, 0.38836,
      0.08362 }, { 0.94977, 0.37729, 0.07905 }, { 0.94538, 0.36638, 0.07461 }, { 0.94084, 0.35566, 0.07031
      }, { 0.93612, 0.34513, 0.06616 }, { 0.93125, 0.33482, 0.06218 }, { 0.92623, 0.32473, 0.05837 }, {
      0.92105, 0.31489, 0.05475 }, { 0.91572, 0.3053, 0.05134 }, { 0.91024, 0.29599, 0.04814 }, { 0.90463,
      0.28696, 0.04516 }, { 0.89888, 0.27824, 0.04243 }, { 0.89298, 0.26981, 0.03993 }, { 0.88691, 0.26152,
      0.03753 }, { 0.88066, 0.25334, 0.03521 }, { 0.87422, 0.24526, 0.03297 }, { 0.8676, 0.2373, 0.03082 },
      { 0.86079, 0.22945, 0.02875 }, { 0.8538, 0.2217, 0.02677 }, { 0.84662, 0.21407, 0.02487 }, { 0.83926,
      0.20654, 0.02305 }, { 0.83172, 0.19912, 0.02131 }, { 0.82399, 0.19182, 0.01966 }, { 0.81608, 0.18462,
      0.01809 }, { 0.80799, 0.17753, 0.0166 }, { 0.79971, 0.17055, 0.0152 }, { 0.79125, 0.16368, 0.01387 },
      { 0.7826, 0.15693, 0.01264 }, { 0.77377, 0.15028, 0.01148 }, { 0.76476, 0.14374, 0.01041 }, { 0.75556,
      0.13731, 0.00942 }, { 0.74617, 0.13098, 0.00851 }, { 0.73661, 0.12477, 0.00769 }, { 0.72686, 0.11867,
      0.00695 }, { 0.71692, 0.11268, 0.00629 }, { 0.7068, 0.1068, 0.00571 }, { 0.6965, 0.10102, 0.00522 }, {
      0.68602, 0.09536, 0.00481 }, { 0.67535, 0.0898, 0.00449 }, { 0.66449, 0.08436, 0.00424 }, { 0.65345,
      0.07902, 0.00408 }, { 0.64223, 0.0738, 0.00401 }, { 0.63082, 0.06868, 0.00401 }, { 0.61923, 0.06367,
      0.0041 }, { 0.60746, 0.05878, 0.00427 }, { 0.5955, 0.05399, 0.00453 }, { 0.58336, 0.04931, 0.00486 },
      { 0.57103, 0.04474, 0.00529 }, { 0.55852, 0.04028, 0.00579 }, { 0.54583, 0.03593, 0.00638 }, {
      0.53295, 0.03169, 0.00705 }, { 0.51989, 0.02756, 0.0078 }, { 0.50664, 0.02354, 0.00863 }, { 0.49321,
      0.01963, 0.00955 }, { 0.4796, 0.01583, 0.01055 } };
00068
00069 /// <summary>
00070 /// Magma gradient, based on the homonymous colour scale included in the "viridis" R package (<seealso
      href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00071 /// </summary>
00072         public static readonly GradientStops Magma = new GradientStops(from el in Enumerable.Range(0,
      21) let ind = (int)Math.Round(12.75 * el) select new GradientStop(Colour.FromRgb(MagmaRGB[ind, 0],
      MagmaRGB[ind, 1], MagmaRGB[ind, 2]), el * 0.05));
00073
00074 /// <summary>
00075 /// Inferno gradient, based on the homonymous colour scale included in the "viridis" R package
      (<seealso href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00076 /// </summary>
00077         public static readonly GradientStops Inferno = new GradientStops(from el in
      Enumerable.Range(0, 21) let ind = (int)Math.Round(12.75 * el) select new
      GradientStop(Colour.FromRgb(InfernoRGB[ind, 0], InfernoRGB[ind, 1], InfernoRGB[ind, 2]), el * 0.05));
00078
00079 /// <summary>
00080 /// Plasma gradient, based on the homonymous colour scale included in the "viridis" R package
      (<seealso href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00081 /// </summary>
00082         public static readonly GradientStops Plasma = new GradientStops(from el in Enumerable.Range(0,
      21) let ind = (int)Math.Round(12.75 * el) select new GradientStop(Colour.FromRgb(PlasmaRGB[ind, 0],
      PlasmaRGB[ind, 1], PlasmaRGB[ind, 2]), el * 0.05));
00083
00084 /// <summary>
00085 /// Viridis gradient, based on the homonymous colour scale included in the "viridis" R package
      (<seealso href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00086 /// </summary>
00087         public static readonly GradientStops Viridis = new GradientStops(from el in
      Enumerable.Range(0, 21) let ind = (int)Math.Round(12.75 * el) select new
      GradientStop(Colour.FromRgb(ViridisRGB[ind, 0], ViridisRGB[ind, 1], ViridisRGB[ind, 2]), el * 0.05));
00088
00089 /// <summary>
00090 /// Cividis gradient, based on the homonymous colour scale included in the "viridis" R package
      (<seealso href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00091 /// </summary>
00092         public static readonly GradientStops Cividis = new GradientStops(from el in
      Enumerable.Range(0, 21) let ind = (int)Math.Round(12.75 * el) select new
      GradientStop(Colour.FromRgb(CividisRGB[ind, 0], CividisRGB[ind, 1], CividisRGB[ind, 2]), el * 0.05));
00093
00094 /// <summary>
00095 /// Rocket gradient, based on the homonymous colour scale included in the "viridis" R package
      (<seealso href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00096 /// </summary>
00097         public static readonly GradientStops Rocket = new GradientStops(from el in Enumerable.Range(0,
      21) let ind = (int)Math.Round(12.75 * el) select new GradientStop(Colour.FromRgb(RocketRGB[ind, 0],
      RocketRGB[ind, 1], RocketRGB[ind, 2]), el * 0.05));
00098
00099 /// <summary>
00100 /// Mako gradient, based on the homonymous colour scale included in the "viridis" R package (<seealso
```

```
      href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00101 /// </summary>
00102         public static readonly GradientStops Mako = new GradientStops(from el in Enumerable.Range(0,
      21) let ind = (int)Math.Round(12.75 * el) select new GradientStop(Colour.FromRgb(MakoRGB[ind, 0],
      MakoRGB[ind, 1], MakoRGB[ind, 2]), el * 0.05));
00103
00104 /// <summary>
00105 /// Turbo gradient, based on the homonymous colour scale included in the "viridis" R package (<seealso
      href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00106 /// </summary>
00107         public static readonly GradientStops Turbo = new GradientStops(from el in Enumerable.Range(0,
      21) let ind = (int)Math.Round(12.75 * el) select new GradientStop(Colour.FromRgb(TurboRGB[ind, 0],
      TurboRGB[ind, 1], TurboRGB[ind, 2]), el * 0.05));
00108
00109 /// <summary>
00110 /// Magma colour map, based on the homonymous colour scale included in the "viridis" R package
      (<seealso href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00111 /// </summary>
00112 /// <param name="x">The position in the map (ranging from 0 to 1).</param>
00113 /// <returns>The colour at the corresponding position in the colour map.</returns>
00114         public static Colour MagmaColouring(double x)
00115         {
00116             x = Math.Max(Math.Min(x, 1), 0);
00117             int ind = (int)Math.Round(x * 255);
00118             return Colour.FromRgb(MagmaRGB[ind, 0], MagmaRGB[ind, 1], MagmaRGB[ind, 2]);
00119         }
00120
00121 /// <summary>
00122 /// Inferno colour map, based on the homonymous colour scale included in the "viridis" R package
      (<seealso href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00123 /// </summary>
00124 /// <param name="x">The position in the map (ranging from 0 to 1).</param>
00125 /// <returns>The colour at the corresponding position in the colour map.</returns>
00126         public static Colour InfernoColouring(double x)
00127         {
00128             x = Math.Max(Math.Min(x, 1), 0);
00129             int ind = (int)Math.Round(x * 255);
00130             return Colour.FromRgb(InfernoRGB[ind, 0], InfernoRGB[ind, 1], InfernoRGB[ind, 2]);
00131         }
00132
00133 /// <summary>
00134 /// Plasma colour map, based on the homonymous colour scale included in the "viridis" R package
      (<seealso href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00135 /// </summary>
00136 /// <param name="x">The position in the map (ranging from 0 to 1).</param>
00137 /// <returns>The colour at the corresponding position in the colour map.</returns>
00138         public static Colour PlasmaColouring(double x)
00139         {
00140             x = Math.Max(Math.Min(x, 1), 0);
00141             int ind = (int)Math.Round(x * 255);
00142             return Colour.FromRgb(PlasmaRGB[ind, 0], PlasmaRGB[ind, 1], PlasmaRGB[ind, 2]);
00143         }
00144
00145 /// <summary>
00146 /// Viridis colour map, based on the homonymous colour scale included in the "viridis" R package
      (<seealso href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00147 /// </summary>
00148 /// <param name="x">The position in the map (ranging from 0 to 1).</param>
00149 /// <returns>The colour at the corresponding position in the colour map.</returns>
00150         public static Colour ViridisColouring(double x)
00151         {
00152             x = Math.Max(Math.Min(x, 1), 0);
00153             int ind = (int)Math.Round(x * 255);
00154             return Colour.FromRgb(ViridisRGB[ind, 0], ViridisRGB[ind, 1], ViridisRGB[ind, 2]);
00155         }
00156
00157 /// <summary>
00158 /// Cividis colour map, based on the homonymous colour scale included in the "viridis" R package
      (<seealso href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00159 /// </summary>
00160 /// <param name="x">The position in the map (ranging from 0 to 1).</param>
00161 /// <returns>The colour at the corresponding position in the colour map.</returns>
00162         public static Colour CividisColouring(double x)
00163         {
00164             x = Math.Max(Math.Min(x, 1), 0);
00165             int ind = (int)Math.Round(x * 255);
00166             return Colour.FromRgb(CividisRGB[ind, 0], CividisRGB[ind, 1], CividisRGB[ind, 2]);
00167         }
00168
00169 /// <summary>
00170 /// Rocket colour map, based on the homonymous colour scale included in the "viridis" R package
      (<seealso href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00171 /// </summary>
00172 /// <param name="x">The position in the map (ranging from 0 to 1).</param>
00173 /// <returns>The colour at the corresponding position in the colour map.</returns>
00174         public static Colour RocketColouring(double x)
00175         {
```

```
00176             x = Math.Max(Math.Min(x, 1), 0);
00177             int ind = (int)Math.Round(x * 255);
00178             return Colour.FromRgb(RocketRGB[ind, 0], RocketRGB[ind, 1], RocketRGB[ind, 2]);
00179         }
00180
00181 /// <summary>
00182 /// Mako colour map, based on the homonymous colour scale included in the "viridis" R package
      (<seealso href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00183 /// </summary>
00184 /// <param name="x">The position in the map (ranging from 0 to 1).</param>
00185 /// <returns>The colour at the corresponding position in the colour map.</returns>
00186         public static Colour MakoColouring(double x)
00187         {
00188             x = Math.Max(Math.Min(x, 1), 0);
00189             int ind = (int)Math.Round(x * 255);
00190             return Colour.FromRgb(MakoRGB[ind, 0], MakoRGB[ind, 1], MakoRGB[ind, 2]);
00191         }
00192
00193 /// <summary>
00194 /// Turbo colour map, based on the homonymous colour scale included in the "viridis" R package
      (<seealso href="https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html"/>).
00195 /// </summary>
00196 /// <param name="x">The position in the map (ranging from 0 to 1).</param>
00197 /// <returns>The colour at the corresponding position in the colour map.</returns>
00198         public static Colour TurboColouring(double x)
00199         {
00200             x = Math.Max(Math.Min(x, 1), 0);
00201             int ind = (int)Math.Round(x * 255);
00202             return Colour.FromRgb(TurboRGB[ind, 0], TurboRGB[ind, 1], TurboRGB[ind, 2]);
00203         }
00204     }
00205 }
```

8.83 Graphics.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Diagnostics.Contracts;
00021 using VectSharp.Filters;
00022
00023 namespace VectSharp
00024 {
00025     internal static class Utils
00026     {
00027         public static Point RotatePoint(Point point, double angle)
00028         {
00029             return new Point(point.X * Math.Cos(angle) - point.Y * Math.Sin(angle), point.X *
      Math.Sin(angle) + point.Y * Math.Cos(angle));
00030         }
00031     }
00032
00033 /// <summary>
00034 /// This interface should be implemented by classes intended to provide graphics output capability to
      a <see cref="Graphics"/> object.
00035 /// </summary>
00036     public interface IGraphicsContext
00037     {
00038 /// <summary>
00039 /// Width of the graphic surface.
00040 /// </summary>
00041         double Width { get; }
00042
00043 /// <summary>
00044 /// Height of the graphic surface.
00045 /// </summary>
00046         double Height { get; }
```

```
00047
00048 /// <summary>
00049 /// Save the current transform state (rotation, translation, scale).  This should be implemented as a
      LIFO stack.
00050 /// </summary>
00051         void Save();
00052
00053 /// <summary>
00054 /// Restore the previous transform state (rotation, translation, scale).  This should be implemented
      as a LIFO stack.
00055 /// </summary>
00056         void Restore();
00057
00058 /// <summary>
00059 /// Translate the coordinate system origin.
00060 /// </summary>
00061 /// <param name="x">The horizontal translation.</param>
00062 /// <param name="y">The vertical translation.</param>
00063         void Translate(double x, double y);
00064
00065 /// <summary>
00066 /// Rotate the coordinate system around the origin.
00067 /// </summary>
00068 /// <param name="angle">The angle (in radians) by which to rotate the coordinate system.</param>
00069         void Rotate(double angle);
00070
00071 /// <summary>
00072 /// Scale the coordinate system with respect to the origin.
00073 /// </summary>
00074 /// <param name="scaleX">The horizontal scale.</param>
00075 /// <param name="scaleY">The vertical scale.</param>
00076         void Scale(double scaleX, double scaleY);
00077
00078 /// <summary>
00079 /// Transform the coordinate system with the specified transformation matrix [ [a, c, e], [b, d, f],
      [0, 0, 1] ].
00080 /// </summary>
00081 /// <param name="a">The first element of the first column.</param>
00082 /// <param name="b">The second element of the first column.</param>
00083 /// <param name="c">The first element of the second column.</param>
00084 /// <param name="d">The second element of the second column.</param>
00085 /// <param name="e">The first element of the third column.</param>
00086 /// <param name="f">The second element of the third column.</param>
00087         void Transform(double a, double b, double c, double d, double e, double f);
00088
00089 /// <summary>
00090 /// The current font.
00091 /// </summary>
00092         Font Font { get; set; }
00093
00094 /// <summary>
00095 /// The current text baseline.
00096 /// </summary>
00097         TextBaselines TextBaseline { get; set; }
00098
00099 /// <summary>
00100 /// Fill a text string using the current <see cref="Font"/> and <see cref="TextBaseline"/>.
00101 /// </summary>
00102 /// <param name="text">The string to draw.</param>
00103 /// <param name="x">The horizontal coordinate of the text origin.</param>
00104 /// <param name="y">The vertical coordinate of the text origin.</param>
00105         void FillText(string text, double x, double y);
00106
00107 /// <summary>
00108 /// Stroke the outline of a text string using the current <see cref="Font"/> and <see
      cref="TextBaseline"/>.
00109 /// </summary>
00110 /// <param name="text">The string to draw.</param>
00111 /// <param name="x">The horizontal coordinate of the text origin.</param>
00112 /// <param name="y">The vertical coordinate of the text origin.</param>
00113         void StrokeText(string text, double x, double y);
00114
00115 /// <summary>
00116 /// Change the current point without drawing a line from the previous point.  If necessary, start a
      new figure.
00117 /// </summary>
00118 /// <param name="x">The horizontal coordinate of the point.</param>
00119 /// <param name="y">The vertical coordinate of the point.</param>
00120         void MoveTo(double x, double y);
00121
00122 /// <summary>
00123 /// Draw a line from the previous point to the specified point.
00124 /// </summary>
00125 /// <param name="x">The horizontal coordinate of the point.</param>
00126 /// <param name="y">The vertical coordinate of the point.</param>
00127         void LineTo(double x, double y);
00128
```

```
00129 /// <summary>
00130 /// Close the current figure.
00131 /// </summary>
00132         void Close();
00133
00134 /// <summary>
00135 /// Stroke the current path using the current <see cref="StrokeStyle"/>, <see cref="LineWidth"/>, <see
      cref="LineCap"/>, <see cref="LineJoin"/> and <see cref="LineDash"/>.
00136 /// </summary>
00137         void Stroke();
00138
00139 /// <summary>
00140 /// Set the current clipping path as the intersection of the previous clipping path and the current
      path.
00141 /// </summary>
00142         void SetClippingPath();
00143
00144 /// <summary>
00145 /// Current brush used to fill paths.
00146 /// </summary>
00147         Brush FillStyle { get; }
00148
00149 /// <summary>
00150 /// Set the current <see cref="FillStyle"/>.
00151 /// </summary>
00152 /// <param name="style">A <see cref="ValueTuple{Int32, Int32, Int32, Double}"/> containing component
      information for the colour.  For r, g, and b, range:  [0, 255]; for a, range:  [0, 1].</param>
00153         void SetFillStyle((int r, int g, int b, double a) style);
00154
00155 /// <summary>
00156 /// Set the current <see cref="FillStyle"/>.
00157 /// </summary>
00158 /// <param name="style">The new fill style.</param>
00159         void SetFillStyle(Brush style);
00160
00161 /// <summary>
00162 /// Current brush used to stroke paths.
00163 /// </summary>
00164         Brush StrokeStyle { get; }
00165
00166 /// <summary>
00167 /// Set the current <see cref="StrokeStyle"/>.
00168 /// </summary>
00169 /// <param name="style">A <see cref="ValueTuple{Int32, Int32, Int32, Double}"/> containing component
      information for the colour.  For r, g, and b, range:  [0, 255]; for a, range:  [0, 1].</param>
00170         void SetStrokeStyle((int r, int g, int b, double a) style);
00171
00172 /// <summary>
00173 /// Set the current <see cref="StrokeStyle"/>.
00174 /// </summary>
00175 /// <param name="style">The new stroke style.</param>
00176         void SetStrokeStyle(Brush style);
00177
00178 /// <summary>
00179 /// Add to the current figure a cubic Bezier from the current point to a destination point, with two
      control points.
00180 /// </summary>
00181 /// <param name="p1X">The horizontal coordinate of the first control point.</param>
00182 /// <param name="p1Y">The vertical coordinate of the first control point.</param>
00183 /// <param name="p2X">The horizontal coordinate of the second control point.</param>
00184 /// <param name="p2Y">The vertical coordinate of the second control point.</param>
00185 /// <param name="p3X">The horizontal coordinate of the destination point.</param>
00186 /// <param name="p3Y">The vertical coordinate of the destination point.</param>
00187         void CubicBezierTo(double p1X, double p1Y, double p2X, double p2Y, double p3X, double p3Y);
00188
00189 /// <summary>
00190 /// Add a rectangle figure to the current path.
00191 /// </summary>
00192 /// <param name="x0">The horizontal coordinate of the top-left corner of the rectangle.</param>
00193 /// <param name="y0">The vertical coordinate of the top-left corner of the rectangle.</param>
00194 /// <param name="width">The width of corner of the rectangle.</param>
00195 /// <param name="height">The height of corner of the rectangle.</param>
00196         void Rectangle(double x0, double y0, double width, double height);
00197
00198 /// <summary>
00199 /// Fill the current path using the current <see cref="FillStyle"/>.
00200 /// </summary>
00201 /// <param name="fillRule">The <see cref="FillRule"/> that determines which parts of the path are
      filled.</param>
00202         void Fill(FillRule fillRule);
00203
00204 /// <summary>
00205 /// Current line width used to stroke paths.
00206 /// </summary>
00207         double LineWidth { get; set; }
00208
00209 /// <summary>
```

```
00210 /// Current line cap used to stroke paths.
00211 /// </summary>
00212         LineCaps LineCap { set; }
00213
00214 /// <summary>
00215 /// Current line join used to stroke paths.
00216 /// </summary>
00217         LineJoins LineJoin { set; }
00218
00219 /// <summary>
00220 /// Set the current line dash pattern.
00221 /// </summary>
00222 /// <param name="dash">The line dash pattern.</param>
00223         void SetLineDash(LineDash dash);
00224
00225 /// <summary>
00226 /// The current tag.  How this can be used depends on each implementation.
00227 /// </summary>
00228         string Tag { get; set; }
00229
00230 /// <summary>
00231 /// Draw a raster image.
00232 /// </summary>
00233 /// <param name="sourceX">The horizontal coordinate of the top-left corner of the rectangle delimiting
      the source area of the image.</param>
00234 /// <param name="sourceY">The vertical coordinate of the top-left corner of the rectangle delimiting
      the source area of the image.</param>
00235 /// <param name="sourceWidth">The width of the rectangle delimiting the source area of the
      image.</param>
00236 /// <param name="sourceHeight">The height of the rectangle delimiting the source area of the
      image.</param>
00237 /// <param name="destinationX">The horizontal coordinate of the top-left corner of the rectangle
      delimiting the destination area of the image.</param>
00238 /// <param name="destinationY">The vertical coordinate of the top-left corner of the rectangle
      delimiting the destination area of the image.</param>
00239 /// <param name="destinationWidth">The width of the rectangle delimiting the destination area of the
      image.</param>
00240 /// <param name="destinationHeight">The height of the rectangle delimiting the destination area of the
      image.</param>
00241 /// <param name="image">The image to draw.</param>
00242         void DrawRasterImage(int sourceX, int sourceY, int sourceWidth, int sourceHeight, double
      destinationX, double destinationY, double destinationWidth, double destinationHeight, RasterImage
      image);
00243
00244 /// <summary>
00245 /// Draws a <see cref="Graphics"/> object, applying the specified <paramref name="filter"/>.
00246 /// </summary>
00247 /// <param name="graphics">The <see cref="Graphics"/> object to draw on the current <see
      cref="Graphics"/> object.</param>
00248 /// <param name="filter">An <see cref="IFilter"/> object, representing the filter to apply to the
      <paramref name="graphics"/> object.</param>
00249         void DrawFilteredGraphics(Graphics graphics, IFilter filter);
00250     }
00251
00252 /// <summary>
00253 /// The exception that is thrown when an unbalanced graphics state stack occurs.
00254 /// </summary>
00255     public class UnbalancedStackException :  Exception
00256     {
00257         internal UnbalancedStackException(int excessSave, int excessRestore) :  base("The graphics
      state stack is unbalanced!\nThere are " + excessSave + " calls to Graphics.Save() and " +
      excessRestore + " calls to Graphics.Restore() in excess!") { }
00258     }
00259
00260 /// <summary>
00261 /// Represents a rule used to determine whether a point is inside or outside of a shape.
00262 /// </summary>
00263     public enum FillRule
00264     {
00265 /// <summary>
00266 /// A point is inside a shape if the perimeter of the shape is hit an odd number of times when moving
      from the point towards the outside.
00267 /// </summary>
00268         EvenOdd,
00269
00270 /// <summary>
00271 /// A point is inside a shape if its winding number is not zero.
00272 /// </summary>
00273         NonZeroWinding
00274     }
00275
00276 /// <summary>
00277 /// Represents an abstract drawing surface.
00278 /// </summary>
00279     public partial class Graphics
00280     {
00281 /// <summary>
```

```
00282 /// Determines how an unbalanced graphics state stack (which occurs if the number of calls to <see
      cref="Save"/> and <see cref="Restore"/> is not equal) will be treated.  The default is <see
      cref="UnbalancedStackActions.Throw"/>.
00283 /// </summary>
00284         public static UnbalancedStackActions UnbalancedStackAction { get; set; } =
      UnbalancedStackActions.Throw;
00285
00286         internal List<IGraphicsAction> Actions = new List<IGraphicsAction>();
00287
00288 /// <summary>
00289 /// The default fill rule.
00290 /// </summary>
00291         public FillRule DefaultFillRule { get; set; } = FillRule.NonZeroWinding;
00292
00293 /// <summary>
00294 /// Fill a <see cref="GraphicsPath"/>.
00295 /// </summary>
00296 /// <param name="path">The <see cref="GraphicsPath"/> to fill.</param>
00297 /// <param name="fillColour">The <see cref="Brush"/> with which to fill the <see
      cref="GraphicsPath"/>.</param>
00298 /// <param name="tag">A tag to identify the filled path.</param>
00299         public void FillPath(GraphicsPath path, Brush fillColour, string tag = null)
00300         {
00301             FillPath(path, fillColour, DefaultFillRule, tag);
00302         }
00303
00304 /// <summary>
00305 /// Fill a <see cref="GraphicsPath"/>.
00306 /// </summary>
00307 /// <param name="path">The <see cref="GraphicsPath"/> to fill.</param>
00308 /// <param name="fillColour">The <see cref="Brush"/> with which to fill the <see
      cref="GraphicsPath"/>.</param>
00309 /// <param name="fillRule">The <see cref="FillRule"/> that determines which parts of the path are
      filled.</param>
00310 /// <param name="tag">A tag to identify the filled path.</param>
00311         public void FillPath(GraphicsPath path, Brush fillColour, FillRule fillRule, string tag =
      null)
00312         {
00313             Actions.Add(new PathAction(path, fillColour, null, 0, LineCaps.Butt, LineJoins.Miter,
      LineDash.SolidLine, tag, fillRule, false));
00314         }
00315
00316 /// <summary>
00317 /// Determines whether unique tags should be used for graphics actions that create multiple objects
      (e.g.  drawing text).
00318 /// </summary>
00319         public bool UseUniqueTags { get; set; } = true;
00320
00321 /// <summary>
00322 /// Stroke a <see cref="GraphicsPath"/>.
00323 /// </summary>
00324 /// <param name="path">The <see cref="GraphicsPath"/> to stroke.</param>
00325 /// <param name="strokeColour">The <see cref="Brush"/> with which to stroke the <see
      cref="GraphicsPath"/>.</param>
00326 /// <param name="lineWidth">The width of the line with which the path is stroked.</param>
00327 /// <param name="lineCap">The line cap to use to stroke the path.</param>
00328 /// <param name="lineJoin">The line join to use to stroke the path.</param>
00329 /// <param name="lineDash">The line dash to use to stroke the path.</param>
00330 /// <param name="tag">A tag to identify the stroked path.</param>
00331         public void StrokePath(GraphicsPath path, Brush strokeColour, double lineWidth = 1, LineCaps
      lineCap = LineCaps.Butt, LineJoins lineJoin = LineJoins.Miter, LineDash?  lineDash = null, string tag
      = null)
00332         {
00333             Actions.Add(new PathAction(path, null, strokeColour, lineWidth, lineCap, lineJoin,
      lineDash ??  LineDash.SolidLine, tag, DefaultFillRule, false));
00334         }
00335
00336 /// <summary>
00337 /// Intersect the current clipping path with the specified <see cref="GraphicsPath"/>.
00338 /// </summary>
00339 /// <param name="path">The <see cref="GraphicsPath"/> to intersect with the current clipping
      path.</param>
00340 /// <param name="tag">A tag to identify the clipping path.</param>
00341         public void SetClippingPath(GraphicsPath path, string tag = null)
00342         {
00343             Actions.Add(new PathAction(path, null, null, 0, LineCaps.Butt, LineJoins.Miter,
      LineDash.SolidLine, tag, DefaultFillRule, true));
00344         }
00345
00346 /// <summary>
00347 /// Intersect the current clipping path with the specified rectangle.
00348 /// </summary>
00349 /// <param name="leftX">The horizontal coordinate of the top-left corner of the rectangle.</param>
00350 /// <param name="topY">The vertical coordinate of the top-left corner of the rectangle.</param>
00351 /// <param name="width">The width of the rectangle.</param>
00352 /// <param name="height">The height of the rectangle.</param>
00353 /// <param name="tag">A tag to identify the clipping path.</param>
```

```
00354         public void SetClippingPath(double leftX, double topY, double width, double height, string tag
      = null)
00355         {
00356             SetClippingPath(new Point(leftX, topY), new Size(width, height), tag);
00357         }
00358 
00359 /// <summary>
00360 /// Intersect the current clipping path with the specified rectangle.
00361 /// </summary>
00362 /// <param name="topLeft">The top-left corner of the rectangle.</param>
00363 /// <param name="size">The size of the rectangle.</param>
00364 /// <param name="tag">A tag to identify the clipping path.</param>
00365         public void SetClippingPath(Point topLeft, Size size, string tag = null)
00366         {
00367             Actions.Add(new PathAction(new GraphicsPath().MoveTo(topLeft).LineTo(topLeft.X +
      size.Width, topLeft.Y).LineTo(topLeft.X + size.Width, topLeft.Y + size.Height).LineTo(topLeft.X,
      topLeft.Y + size.Height).Close(), null, null, 0, LineCaps.Butt, LineJoins.Miter, LineDash.SolidLine,
      tag, DefaultFillRule, true));
00368         }
00369 
00370 /// <summary>
00371 /// Rotate the coordinate system around the origin.
00372 /// </summary>
00373 /// <param name="angle">The angle (in radians) by which to rotate the coordinate system.</param>
00374 /// <param name="tag">A tag to identify the transform.</param>
00375         public void Rotate(double angle, string tag = null)
00376         {
00377             Actions.Add(new TransformAction(angle, tag));
00378         }
00379 
00380 /// <summary>
00381 /// Rotate the coordinate system around a pivot point.
00382 /// </summary>
00383 /// <param name="angle">The angle (in radians) by which to rotate the coordinate system.</param>
00384 /// <param name="pivot">The pivot around which the coordinate system is to be rotated.</param>
00385 /// <param name="tag">A tag to identify the transform.</param>
00386         public void RotateAt(double angle, Point pivot, string tag = null)
00387         {
00388             string tag1 = null;
00389             string tag2 = null;
00390             string tag3 = null;
00391 
00392             if (!string.IsNullOrEmpty(tag))
00393             {
00394                 if (UseUniqueTags)
00395                 {
00396                     tag1 = tag + "_@1";
00397                     tag2 = tag + "_@2";
00398                     tag3 = tag + "_@3";
00399                 }
00400                 else
00401                 {
00402                     tag1 = tag;
00403                     tag2 = tag;
00404                     tag3 = tag;
00405                 }
00406             }
00407 
00408             Actions.Add(new TransformAction(pivot, tag1));
00409             Actions.Add(new TransformAction(angle, tag2));
00410             Actions.Add(new TransformAction(new Point(-pivot.X, -pivot.Y), tag3));
00411         }
00412 
00413 
00414 /// <summary>
00415 /// Transform the coordinate system with the specified transformation matrix [ [a, c, e], [b, d, f],
      [0, 0, 1] ].
00416 /// </summary>
00417 /// <param name="a">The first element of the first column.</param>
00418 /// <param name="b">The second element of the first column.</param>
00419 /// <param name="c">The first element of the second column.</param>
00420 /// <param name="d">The second element of the second column.</param>
00421 /// <param name="e">The first element of the third column.</param>
00422 /// <param name="f">The second element of the third column.</param>
00423 /// <param name="tag">A tag to identify the transform.</param>
00424         public void Transform(double a, double b, double c, double d, double e, double f, string tag =
      null)
00425         {
00426             double[,] matrix = new double[,] { { a, c, e }, { b, d, f }, { 0, 0, 1 } };
00427             Actions.Add(new TransformAction(matrix, tag));
00428         }
00429 
00430 /// <summary>
00431 /// Translate the coordinate system origin.
00432 /// </summary>
00433 /// <param name="x">The horizontal translation.</param>
00434 /// <param name="y">The vertical translation.</param>
```

```
00435 /// <param name="tag">A tag to identify the transform.</param>
00436         public void Translate(double x, double y, string tag = null)
00437         {
00438             Actions.Add(new TransformAction(new Point(x, y), tag));
00439         }
00440 
00441 /// <summary>
00442 /// Translate the coordinate system origin.
00443 /// </summary>
00444 /// <param name="delta">The new origin point.</param>
00445 /// <param name="tag">A tag to identify the transform.</param>
00446         public void Translate(Point delta, string tag = null)
00447         {
00448             Actions.Add(new TransformAction(delta, tag));
00449         }
00450 
00451 /// <summary>
00452 /// Scale the coordinate system with respect to the origin.
00453 /// </summary>
00454 /// <param name="scaleX">The horizontal scale.</param>
00455 /// <param name="scaleY">The vertical scale.</param>
00456 /// <param name="tag">A tag to identify the transform.</param>
00457         public void Scale(double scaleX, double scaleY, string tag = null)
00458         {
00459             Actions.Add(new TransformAction(new Size(scaleX, scaleY), tag));
00460         }
00461 
00462 /// <summary>
00463 /// Fill a rectangle.
00464 /// </summary>
00465 /// <param name="topLeft">The top-left corner of the rectangle.</param>
00466 /// <param name="size">The size of the rectangle.</param>
00467 /// <param name="fillColour">The colour with which to fill the rectangle.</param>
00468 /// <param name="tag">A tag to identify the filled rectangle.</param>
00469         public void FillRectangle(Point topLeft, Size size, Brush fillColour, string tag = null)
00470         {
00471             Actions.Add(new RectangleAction(topLeft, size, fillColour, null, 0, LineCaps.Butt,
      LineJoins.Miter, LineDash.SolidLine, tag));
00472         }
00473 
00474 /// <summary>
00475 /// Fill a rectangle.
00476 /// </summary>
00477 /// <param name="leftX">The horizontal coordinate of the top-left corner of the rectangle.</param>
00478 /// <param name="topY">The vertical coordinate of the top-left corner of the rectangle.</param>
00479 /// <param name="width">The width of the rectangle.</param>
00480 /// <param name="height">The height of the rectangle.</param>
00481 /// <param name="fillColour">The colour with which to fill the rectangle.</param>
00482 /// <param name="tag">A tag to identify the filled rectangle.</param>
00483         public void FillRectangle(double leftX, double topY, double width, double height, Brush
      fillColour, string tag = null)
00484         {
00485             Actions.Add(new RectangleAction(new Point(leftX, topY), new Size(width, height),
      fillColour, null, 0, LineCaps.Butt, LineJoins.Miter, LineDash.SolidLine, tag));
00486         }
00487 
00488 /// <summary>
00489 /// Stroke a rectangle.
00490 /// </summary>
00491 /// <param name="topLeft">The top-left corner of the rectangle.</param>
00492 /// <param name="size">The size of the rectangle.</param>
00493 /// <param name="strokeColour">The colour with which to stroke the rectangle.</param>
00494 /// <param name="lineWidth">The width of the line with which the rectangle is stroked.</param>
00495 /// <param name="lineCap">The line cap to use to stroke the rectangle.</param>
00496 /// <param name="lineJoin">The line join to use to stroke the rectangle.</param>
00497 /// <param name="lineDash">The line dash to use to stroke the rectangle.</param>
00498 /// <param name="tag">A tag to identify the filled rectangle.</param>
00499         public void StrokeRectangle(Point topLeft, Size size, Brush strokeColour, double lineWidth =
      1, LineCaps lineCap = LineCaps.Butt, LineJoins lineJoin = LineJoins.Miter, LineDash?  lineDash = null,
      string tag = null)
00500         {
00501             Actions.Add(new RectangleAction(topLeft, size, null, strokeColour, lineWidth, lineCap,
      lineJoin, lineDash ??  LineDash.SolidLine, tag));
00502         }
00503 
00504 /// <summary>
00505 /// Stroke a rectangle.
00506 /// </summary>
00507 /// <param name="leftX">The horizontal coordinate of the top-left corner of the rectangle.</param>
00508 /// <param name="topY">The vertical coordinate of the top-left corner of the rectangle.</param>
00509 /// <param name="width">The width of the rectangle.</param>
00510 /// <param name="height">The height of the rectangle.</param>
00511 /// <param name="strokeColour">The colour with which to stroke the rectangle.</param>
00512 /// <param name="lineWidth">The width of the line with which the rectangle is stroked.</param>
00513 /// <param name="lineCap">The line cap to use to stroke the rectangle.</param>
00514 /// <param name="lineJoin">The line join to use to stroke the rectangle.</param>
00515 /// <param name="lineDash">The line dash to use to stroke the rectangle.</param>
```

```
00516 /// <param name="tag">A tag to identify the filled rectangle.</param>
00517         public void StrokeRectangle(double leftX, double topY, double width, double height, Brush
      strokeColour, double lineWidth = 1, LineCaps lineCap = LineCaps.Butt, LineJoins lineJoin =
      LineJoins.Miter, LineDash?  lineDash = null, string tag = null)
00518         {
00519             Actions.Add(new RectangleAction(new Point(leftX, topY), new Size(width, height), null,
      strokeColour, lineWidth, lineCap, lineJoin, lineDash ??  LineDash.SolidLine, tag));
00520         }
00521
00522 /// <summary>
00523 /// Draw a raster image.
00524 /// </summary>
00525 /// <param name="sourceX">The horizontal coordinate of the top-left corner of the rectangle delimiting
      the source area of the image.</param>
00526 /// <param name="sourceY">The vertical coordinate of the top-left corner of the rectangle delimiting
      the source area of the image.</param>
00527 /// <param name="sourceWidth">The width of the rectangle delimiting the source area of the
      image.</param>
00528 /// <param name="sourceHeight">The height of the rectangle delimiting the source area of the
      image.</param>
00529 /// <param name="destinationX">The horizontal coordinate of the top-left corner of the rectangle
      delimiting the destination area of the image.</param>
00530 /// <param name="destinationY">The vertical coordinate of the top-left corner of the rectangle
      delimiting the destination area of the image.</param>
00531 /// <param name="destinationWidth">The width of the rectangle delimiting the destination area of the
      image.</param>
00532 /// <param name="destinationHeight">The height of the rectangle delimiting the destination area of the
      image.</param>
00533 /// <param name="image">The image to draw.</param>
00534 /// <param name="tag">A tag to identify the drawn image.</param>
00535         public void DrawRasterImage(int sourceX, int sourceY, int sourceWidth, int sourceHeight,
      double destinationX, double destinationY, double destinationWidth, double destinationHeight,
      RasterImage image, string tag = null)
00536         {
00537             Actions.Add(new RasterImageAction(sourceX, sourceY, sourceWidth, sourceHeight,
      destinationX, destinationY, destinationWidth, destinationHeight, image, tag));
00538         }
00539
00540 /// <summary>
00541 /// Draw a raster image.
00542 /// </summary>
00543 /// <param name="x">The horizontal coordinate of the top-left corner of the rectangle delimiting the
      destination area of the image.</param>
00544 /// <param name="y">The vertical coordinate of the top-left corner of the rectangle delimiting the
      destination area of the image.</param>
00545 /// <param name="image">The image to draw.</param>
00546 /// <param name="tag">A tag to identify the drawn image.</param>
00547         public void DrawRasterImage(double x, double y, RasterImage image, string tag = null)
00548         {
00549             DrawRasterImage(0, 0, image.Width, image.Height, x, y, image.Width, image.Height, image,
      tag);
00550         }
00551
00552 /// <summary>
00553 /// Draw a raster image.
00554 /// </summary>
00555 /// <param name="position">The the top-left corner of the rectangle delimiting the destination area of
      the image.</param>
00556 /// <param name="image">The image to draw.</param>
00557 /// <param name="tag">A tag to identify the drawn image.</param>
00558         public void DrawRasterImage(Point position, RasterImage image, string tag = null)
00559         {
00560             DrawRasterImage(0, 0, image.Width, image.Height, position.X, position.Y, image.Width,
      image.Height, image, tag);
00561         }
00562
00563 /// <summary>
00564 /// Draw a raster image.
00565 /// </summary>
00566 /// <param name="x">The horizontal coordinate of the top-left corner of the rectangle delimiting the
      destination area of the image.</param>
00567 /// <param name="y">The vertical coordinate of the top-left corner of the rectangle delimiting the
      destination area of the image.</param>
00568 /// <param name="width">The width of the rectangle delimiting the destination area of the
      image.</param>
00569 /// <param name="height">The height of the rectangle delimiting the destination area of the
      image.</param>
00570 /// <param name="image">The image to draw.</param>
00571 /// <param name="tag">A tag to identify the drawn image.</param>
00572         public void DrawRasterImage(double x, double y, double width, double height, RasterImage
      image, string tag = null)
00573         {
00574             DrawRasterImage(0, 0, image.Width, image.Height, x, y, width, height, image, tag);
00575         }
00576
00577 /// <summary>
00578 /// Draw a raster image.
```

```
00579 /// </summary>
00580 /// <param name="position">The the top-left corner of the rectangle delimiting the destination area of
      the image.</param>
00581 /// <param name="size">The size of the rectangle delimiting the destination area of the image.</param>
00582 /// <param name="image">The image to draw.</param>
00583 /// <param name="tag">A tag to identify the drawn image.</param>
00584         public void DrawRasterImage(Point position, Size size, RasterImage image, string tag = null)
00585         {
00586             DrawRasterImage(0, 0, image.Width, image.Height, position.X, position.Y, size.Width,
      size.Height, image, tag);
00587         }
00588 
00589 /// <summary>
00590 /// Save the current transform state (rotation, translation, scale).
00591 /// </summary>
00592         public void Save()
00593         {
00594             Actions.Add(new StateAction(StateAction.StateActionTypes.Save));
00595         }
00596 
00597 /// <summary>
00598 /// Restore the previous transform state (rotation, translation scale).
00599 /// </summary>
00600         public void Restore()
00601         {
00602             Actions.Add(new StateAction(StateAction.StateActionTypes.Restore));
00603         }
00604 
00605         private void FixGraphicsStateStack()
00606         {
00607             if (UnbalancedStackAction == UnbalancedStackActions.Ignore)
00608             {
00609                 return;
00610             }
00611 
00612             int count = 0;
00613             List<int> toBeRemoved = new List<int>();
00614 
00615             for (int i = 0; i < this.Actions.Count; i++)
00616             {
00617                 if (this.Actions[i] is StateAction st)
00618                 {
00619                     if (st.StateActionType == StateAction.StateActionTypes.Save)
00620                     {
00621                         count++;
00622                     }
00623                     else if (st.StateActionType == StateAction.StateActionTypes.Restore)
00624                     {
00625                         if (count == 0)
00626                         {
00627                             toBeRemoved.Add(i);
00628                         }
00629                         else
00630                         {
00631                             count--;
00632                         }
00633                     }
00634                 }
00635             }
00636 
00637             if (UnbalancedStackAction == UnbalancedStackActions.Throw)
00638             {
00639                 if (count > 0 || toBeRemoved.Count > 0)
00640                 {
00641                     throw new UnbalancedStackException(count, toBeRemoved.Count);
00642                 }
00643             }
00644             else if (UnbalancedStackAction == UnbalancedStackActions.SilentlyFix)
00645             {
00646                 if (count > 0 || toBeRemoved.Count > 0)
00647                 {
00648                     for (int i = toBeRemoved.Count - 1; i >= 0; i--)
00649                     {
00650                         this.Actions.RemoveAt(toBeRemoved[i]);
00651                     }
00652 
00653                     for (int i = 0; i < count; i++)
00654                     {
00655                         this.Restore();
00656                     }
00657 
00658                     this.FixGraphicsStateStack();
00659                 }
00660             }
00661         }
00662 
00663 /// <summary>
```

```
00664 /// Copy the current graphics to an instance of a class implementing <see cref="IGraphicsContext"/>.
00665 /// </summary>
00666 /// <param name="destinationContext">The <see cref="IGraphicsContext"/> on which the graphics are to
      be copied.</param>
00667         public void CopyToIGraphicsContext(IGraphicsContext destinationContext)
00668         {
00669             this.FixGraphicsStateStack();
00670 
00671             for (int i = 0; i < this.Actions.Count; i++)
00672             {
00673                 if (this.Actions[i] is RectangleAction)
00674                 {
00675                     RectangleAction rec = this.Actions[i] as RectangleAction;
00676 
00677                     destinationContext.Tag = rec.Tag;
00678                     destinationContext.Rectangle(rec.TopLeft.X, rec.TopLeft.Y, rec.Size.Width,
      rec.Size.Height);
00679 
00680                     if (rec.Fill != null)
00681                     {
00682                         if (destinationContext.FillStyle != rec.Fill)
00683                         {
00684                             destinationContext.SetFillStyle(rec.Fill);
00685                         }
00686                         destinationContext.Fill(FillRule.EvenOdd);
00687                     }
00688                     else if (rec.Stroke != null)
00689                     {
00690                         if (destinationContext.StrokeStyle != rec.Stroke)
00691                         {
00692                             destinationContext.SetStrokeStyle(rec.Stroke);
00693                         }
00694                         if (destinationContext.LineWidth != rec.LineWidth)
00695                         {
00696                             destinationContext.LineWidth = rec.LineWidth;
00697                         }
00698                         destinationContext.SetLineDash(rec.LineDash);
00699                         destinationContext.LineCap = rec.LineCap;
00700                         destinationContext.LineJoin = rec.LineJoin;
00701 
00702                         destinationContext.Stroke();
00703                     }
00704                 }
00705                 else if (this.Actions[i] is PathAction)
00706                 {
00707                     PathAction pth = this.Actions[i] as PathAction;
00708 
00709                     destinationContext.Tag = pth.Tag;
00710 
00711                     for (int j = 0; j < pth.Path.Segments.Count; j++)
00712                     {
00713                         switch (pth.Path.Segments[j].Type)
00714                         {
00715                             case SegmentType.Move:
00716                                 destinationContext.MoveTo(pth.Path.Segments[j].Point.X,
      pth.Path.Segments[j].Point.Y);
00717                                 break;
00718                             case SegmentType.Line:
00719                                 destinationContext.LineTo(pth.Path.Segments[j].Point.X,
      pth.Path.Segments[j].Point.Y);
00720                                 break;
00721                             case SegmentType.CubicBezier:
00722                                 destinationContext.CubicBezierTo(pth.Path.Segments[j].Points[0].X,
      pth.Path.Segments[j].Points[0].Y, pth.Path.Segments[j].Points[1].X, pth.Path.Segments[j].Points[1].Y,
      pth.Path.Segments[j].Points[2].X, pth.Path.Segments[j].Points[2].Y);
00723                                 break;
00724                             case SegmentType.Arc:
00725                                 {
00726                                     ArcSegment seg = pth.Path.Segments[j] as ArcSegment;
00727                                     Segment[] segs = seg.ToBezierSegments();
00728                                     for (int k = 0; k < segs.Length; k++)
00729                                     {
00730                                         switch (segs[k].Type)
00731                                         {
00732                                             case SegmentType.Move:
00733                                                 destinationContext.MoveTo(segs[k].Point.X,
      segs[k].Point.Y);
00734                                                 break;
00735                                             case SegmentType.Line:
00736                                                 destinationContext.LineTo(segs[k].Point.X,
      segs[k].Point.Y);
00737                                                 break;
00738                                             case SegmentType.CubicBezier:
00739                                                 destinationContext.CubicBezierTo(segs[k].Points[0].X,
      segs[k].Points[0].Y, segs[k].Points[1].X, segs[k].Points[1].Y, segs[k].Points[2].X,
      segs[k].Points[2].Y);
00740                                                 break;
```

```
00741                                       }
00742                                   }
00743                               }
00744                               break;
00745                           case SegmentType.Close:
00746                               destinationContext.Close();
00747                               break;
00748                       }
00749                   }
00750
00751                   if (pth.IsClipping)
00752                   {
00753                       destinationContext.SetClippingPath();
00754                   }
00755                   else
00756                   {
00757                       if (pth.Fill != null)
00758                       {
00759                           if (destinationContext.FillStyle != pth.Fill)
00760                           {
00761                               destinationContext.SetFillStyle(pth.Fill);
00762                           }
00763                           destinationContext.Fill(pth.FillRule);
00764                       }
00765                       else if (pth.Stroke != null)
00766                       {
00767                           if (destinationContext.StrokeStyle != pth.Stroke)
00768                           {
00769                               destinationContext.SetStrokeStyle(pth.Stroke);
00770                           }
00771                           if (destinationContext.LineWidth != pth.LineWidth)
00772                           {
00773                               destinationContext.LineWidth = pth.LineWidth;
00774                           }
00775                           destinationContext.SetLineDash(pth.LineDash);
00776                           destinationContext.LineCap = pth.LineCap;
00777                           destinationContext.LineJoin = pth.LineJoin;
00778
00779                           destinationContext.Stroke();
00780                       }
00781                   }
00782               }
00783               else if (this.Actions[i] is TextAction)
00784               {
00785                   TextAction txt = this.Actions[i] as TextAction;
00786
00787                   destinationContext.Tag = txt.Tag;
00788                   if (destinationContext.TextBaseline != txt.TextBaseline)
00789                   {
00790                       destinationContext.TextBaseline = txt.TextBaseline;
00791                   }
00792                   destinationContext.Font = txt.Font;
00793
00794                   if (txt.Fill != null)
00795                   {
00796                       if (destinationContext.FillStyle != txt.Fill)
00797                       {
00798                           destinationContext.SetFillStyle(txt.Fill);
00799                       }
00800                       destinationContext.FillText(txt.Text, txt.Origin.X, txt.Origin.Y);
00801                   }
00802                   else if (txt.Stroke != null)
00803                   {
00804                       if (destinationContext.StrokeStyle != txt.Stroke)
00805                       {
00806                           destinationContext.SetStrokeStyle(txt.Stroke);
00807                       }
00808                       if (destinationContext.LineWidth != txt.LineWidth)
00809                       {
00810                           destinationContext.LineWidth = txt.LineWidth;
00811                       }
00812                       destinationContext.SetLineDash(txt.LineDash);
00813                       destinationContext.LineCap = txt.LineCap;
00814                       destinationContext.LineJoin = txt.LineJoin;
00815
00816                       destinationContext.StrokeText(txt.Text, txt.Origin.X, txt.Origin.Y);
00817                   }
00818               }
00819               else if (this.Actions[i] is TransformAction)
00820               {
00821                   TransformAction trf = this.Actions[i] as TransformAction;
00822
00823                   destinationContext.Tag = trf.Tag;
00824
00825                   if (trf.Delta != null)
00826                   {
00827                       destinationContext.Translate(((Point)trf.Delta).X, ((Point)trf.Delta).Y);
```

```
00828                     }
00829                     else if (trf.Angle != null)
00830                     {
00831                         destinationContext.Rotate((double)trf.Angle);
00832                     }
00833                     else if (trf.Scale != null)
00834                     {
00835                         destinationContext.Scale(((Size)trf.Scale).Width, ((Size)trf.Scale).Height);
00836                     }
00837                     else if (trf.Matrix != null)
00838                     {
00839                         destinationContext.Transform(trf.Matrix[0, 0], trf.Matrix[1, 0], trf.Matrix[0,
      1], trf.Matrix[1, 1], trf.Matrix[0, 2], trf.Matrix[1, 2]);
00840                     }
00841                 }
00842                 else if (this.Actions[i] is StateAction)
00843                 {
00844                     destinationContext.Tag = ((StateAction)this.Actions[i]).Tag;
00845 
00846                     if (((StateAction)this.Actions[i]).StateActionType ==
      StateAction.StateActionTypes.Save)
00847                     {
00848                         destinationContext.Save();
00849                     }
00850                     else
00851                     {
00852                         destinationContext.Restore();
00853                     }
00854                 }
00855                 else if (this.Actions[i] is RasterImageAction)
00856                 {
00857                     RasterImageAction img = this.Actions[i] as RasterImageAction;
00858                     destinationContext.Tag = img.Tag;
00859                     destinationContext.DrawRasterImage(img.SourceX, img.SourceY, img.SourceWidth,
      img.SourceHeight, img.DestinationX, img.DestinationY, img.DestinationWidth, img.DestinationHeight,
      img.Image);
00860                 }
00861                 else if (this.Actions[i] is FilteredGraphicsAction)
00862                 {
00863                     FilteredGraphicsAction fil = this.Actions[i] as FilteredGraphicsAction;
00864                     destinationContext.Tag = fil.Tag;
00865                     destinationContext.DrawFilteredGraphics(fil.Content, fil.Filter);
00866                 }
00867             }
00868         }
00869 
00870 /// <summary>
00871 /// Draws a <see cref="Graphics"/> object on the current <see cref="Graphics"/> object.
00872 /// </summary>
00873 /// <param name="origin">The point at which to place the origin of <paramref
      name="graphics"/>.</param>
00874 /// <param name="graphics">The <see cref="Graphics"/> object to draw on the current <see
      cref="Graphics"/> object.</param>
00875         public void DrawGraphics(Point origin, Graphics graphics)
00876         {
00877             this.Save();
00878             this.Translate(origin);
00879 
00880             graphics.FixGraphicsStateStack();
00881 
00882             this.Actions.AddRange(graphics.Actions);
00883 
00884             this.Restore();
00885         }
00886 
00887 /// <summary>
00888 /// Draws a <see cref="Graphics"/> object on the current <see cref="Graphics"/> object, prepending the
      supplied <paramref name="tag"/> to the tags contained in the <see cref="Graphics"/> object being
      drawn.
00889 /// </summary>
00890 /// <param name="originX">The horizontal coordinate at which to place the origin of <paramref
      name="graphics"/>.</param>
00891 /// <param name="originY">The vertical coordinate at which to place the origin of <paramref
      name="graphics"/>.</param>
00892 /// <param name="graphics">The <see cref="Graphics"/> object to draw on the current <see
      cref="Graphics"/> object.</param>
00893 /// <param name="tag">The tag to prepend to the tags contained in the <paramref name="graphics"/>
      object.</param>
00894         public void DrawGraphics(double originX, double originY, Graphics graphics, string tag)
00895         {
00896             this.DrawGraphics(new Point(originX, originY), graphics, tag);
00897         }
00898 
00899 /// <summary>
00900 /// Draws a <see cref="Graphics"/> object on the current <see cref="Graphics"/> object, prepending the
      supplied <paramref name="tag"/> to the tags contained in the <see cref="Graphics"/> object being
      drawn.
```

```
00901 /// </summary>
00902 /// <param name="origin">The point at which to place the origin of <paramref
      name="graphics"/>.</param>
00903 /// <param name="graphics">The <see cref="Graphics"/> object to draw on the current <see
      cref="Graphics"/> object.</param>
00904 /// <param name="tag">The tag to prepend to the tags contained in the <paramref name="graphics"/>
      object.</param>
00905         public void DrawGraphics(Point origin, Graphics graphics, string tag)
00906         {
00907             this.Save();
00908 
00909             if (!string.IsNullOrEmpty(tag) && UseUniqueTags)
00910             {
00911                 this.Translate(origin, tag + "@t");
00912             }
00913             else
00914             {
00915                 this.Translate(origin, tag);
00916             }
00917 
00918             graphics.FixGraphicsStateStack();
00919 
00920             foreach (IGraphicsAction action in graphics.Actions)
00921             {
00922                 IGraphicsAction clone = action.ShallowClone();
00923 
00924                 if (clone is IPrintableAction print)
00925                 {
00926                     if (UseUniqueTags)
00927                     {
00928                         if (string.IsNullOrEmpty(print.Tag))
00929                         {
00930                             print.Tag = tag;
00931                         }
00932                         else
00933                         {
00934                             print.Tag = tag + "/" + print.Tag;
00935                         }
00936                     }
00937                 }
00938 
00939                 this.Actions.Add(clone);
00940             }
00941 
00942             this.Restore();
00943         }
00944 
00945 /// <summary>
00946 /// Draws a <see cref="Graphics"/> object on the current <see cref="Graphics"/> object.
00947 /// </summary>
00948 /// <param name="originX">The horizontal coordinate at which to place the origin of <paramref
      name="graphics"/>.</param>
00949 /// <param name="originY">The vertical coordinate at which to place the origin of <paramref
      name="graphics"/>.</param>
00950 /// <param name="graphics">The <see cref="Graphics"/> object to draw on the current <see
      cref="Graphics"/> object.</param>
00951         public void DrawGraphics(double originX, double originY, Graphics graphics)
00952         {
00953             this.DrawGraphics(new Point(originX, originY), graphics);
00954         }
00955 
00956 /// <summary>
00957 /// Draws a <see cref="Graphics"/> object on the current <see cref="Graphics"/> object, applying the
      specified <paramref name="filter"/>.
00958 /// </summary>
00959 /// <param name="origin">The point at which to place the origin of <paramref
      name="graphics"/>.</param>
00960 /// <param name="graphics">The <see cref="Graphics"/> object to draw on the current <see
      cref="Graphics"/> object.</param>
00961 /// <param name="filter">An <see cref="IFilter"/> object, representing the filter to apply to the
      <paramref name="graphics"/> object.</param>
00962 /// <param name="tag">A tag to identify the filter.</param>
00963         public void DrawGraphics(Point origin, Graphics graphics, IFilter filter, string tag = null)
00964         {
00965             this.Save();
00966 
00967             if (!string.IsNullOrEmpty(tag) && UseUniqueTags)
00968             {
00969                 this.Translate(origin, tag + "@t");
00970             }
00971             else
00972             {
00973                 this.Translate(origin, tag);
00974             }
00975 
00976             Graphics clone = new Graphics();
00977             clone.Actions.AddRange(graphics.Actions);
```

```
00978             clone.FixGraphicsStateStack();
00979
00980             this.Actions.Add(new FilteredGraphicsAction(clone, filter) { Tag = tag });
00981
00982             this.Restore();
00983         }
00984
00985 /// <summary>
00986 /// Draws a <see cref="Graphics"/> object on the current <see cref="Graphics"/> object, applying the
      specified <paramref name="filter"/>.
00987 /// </summary>
00988 /// <param name="originX">The horizontal coordinate at which to place the origin of <paramref
      name="graphics"/>.</param>
00989 /// <param name="originY">The vertical coordinate at which to place the origin of <paramref
      name="graphics"/>.</param>
00990 /// <param name="graphics">The <see cref="Graphics"/> object to draw on the current <see
      cref="Graphics"/> object.</param>
00991 /// <param name="filter">An <see cref="IFilter"/> object, representing the filter to apply to the
      <paramref name="graphics"/> object.</param>
00992 /// <param name="tag">A tag to identify the filter.</param>
00993         public void DrawGraphics(double originX, double originY, Graphics graphics, IFilter filter,
      string tag = null)
00994         {
00995             this.DrawGraphics(new Point(originX, originY), graphics, filter, tag);
00996         }
00997
00998
00999         internal static Point Multiply(double[,] matrix, Point pt)
01000         {
01001             double x = matrix[0, 0] * pt.X + matrix[0, 1] * pt.Y + matrix[0, 2];
01002             double y = matrix[1, 0] * pt.X + matrix[1, 1] * pt.Y + matrix[1, 2];
01003             double z = matrix[2, 0] * pt.X + matrix[2, 1] * pt.Y + matrix[2, 2];
01004
01005             return new Point(x / z, y / z);
01006         }
01007
01008         internal static double[,] Multiply(double[,] m1, double[,] m2)
01009         {
01010             return new double[3, 3]
01011             {
01012                 { m1[0,0] * m2[0,0] + m1[0,1] * m2[1,0] + m1[0,2] *m2[2,0], m1[0,0] * m2[0,1] +
      m1[0,1] * m2[1,1] + m1[0,2] * m2[2,1], m1[0,0] * m2[0,2] + m1[0,1] * m2[1,2] + m1[0,2] * m2[2,2] },
01013                 { m1[1,0] * m2[0,0] + m1[1,1] * m2[1,0] + m1[1,2] *m2[2,0], m1[1,0] * m2[0,1] +
      m1[1,1] * m2[1,1] + m1[1,2] * m2[2,1], m1[1,0] * m2[0,2] + m1[1,1] * m2[1,2] + m1[1,2] * m2[2,2] },
01014                 { m1[2,0] * m2[0,0] + m1[2,1] * m2[1,0] + m1[2,2] *m2[2,0], m1[2,0] * m2[0,1] +
      m1[2,1] * m2[1,1] + m1[2,2] * m2[2,1], m1[2,0] * m2[0,2] + m1[2,1] * m2[1,2] + m1[2,2] * m2[2,2] }
01015             };
01016         }
01017
01018         internal static double[,] TranslationMatrix(double dx, double dy)
01019         {
01020             return new double[3, 3]
01021             {
01022                 {1, 0, dx },
01023                 {0, 1, dy },
01024                 {0, 0, 1 }
01025             };
01026         }
01027
01028         internal static double[,] ScaleMatrix(double sx, double sy)
01029         {
01030             return new double[3, 3]
01031             {
01032                 {sx, 0, 0 },
01033                 {0, sy, 0 },
01034                 {0, 0, 1 }
01035             };
01036         }
01037
01038         internal static double[,] RotationMatrix(double theta)
01039         {
01040             return new double[3, 3]
01041             {
01042                 {Math.Cos(theta), -Math.Sin(theta), 0 },
01043                 {Math.Sin(theta), Math.Cos(theta), 0 },
01044                 {0, 0, 1 }
01045             };
01046         }
01047
01048         internal static double[,] Invert(double[,] m)
01049         {
01050             double[,] tbr = new double[3, 3];
01051
01052             tbr[0, 0] = (m[1, 1] * m[2, 2] - m[1, 2] * m[2, 1]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
      0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
      * m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
01053             tbr[0, 1] = -(m[0, 1] * m[2, 2] - m[0, 2] * m[2, 1]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
```

```
0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
* m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
01054             tbr[0, 2] = (m[0, 1] * m[1, 2] - m[0, 2] * m[1, 1]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
* m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
01055             tbr[1, 0] = -(m[1, 0] * m[2, 2] - m[1, 2] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
* m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
01056             tbr[1, 1] = (m[0, 0] * m[2, 2] - m[0, 2] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
* m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
01057             tbr[1, 2] = -(m[0, 0] * m[1, 2] - m[0, 2] * m[1, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
* m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
01058             tbr[2, 0] = (m[1, 0] * m[2, 1] - m[1, 1] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
* m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
01059             tbr[2, 1] = -(m[0, 0] * m[2, 1] - m[0, 1] * m[2, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
* m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
01060             tbr[2, 2] = (m[0, 0] * m[1, 1] - m[0, 1] * m[1, 0]) / (m[0, 0] * m[1, 1] * m[2, 2] - m[0,
0] * m[1, 2] * m[2, 1] - m[1, 0] * m[0, 1] * m[2, 2] + m[2, 0] * m[0, 1] * m[1, 2] + m[1, 0] * m[0, 2]
* m[2, 1] - m[2, 0] * m[0, 2] * m[1, 1]);
01061
01062             return tbr;
01063         }
01064
01065 /// <summary>
01066 /// Creates a new <see cref="Graphics"/> object in which all the graphics actions have been
transformed using an arbitrary transformation function.  Raster images are replaced by grey
rectangles.
01067 /// </summary>
01068 /// <param name="transformationFunction">An arbitrary transformation function.</param>
01069 /// <param name="linearisationResolution">The resolution that will be used to linearise curve
segments.</param>
01070 /// <returns>A new <see cref="Graphics"/> object in which all graphics actions have been linearised
and transformed using the <paramref name="transformationFunction"/>.</returns>
01071         [Pure]
01072         public Graphics Transform(Func<Point, Point> transformationFunction, double
linearisationResolution)
01073         {
01074             Graphics destinationGraphics = new Graphics();
01075
01076             Stack<double[,]> transformMatrix = new Stack<double[,]>();
01077             double[,] currMatrix = new double[3, 3] { { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0, 1 } };
01078
01079             for (int i = 0; i < this.Actions.Count; i++)
01080             {
01081                 if (this.Actions[i] is RectangleAction)
01082                 {
01083                     RectangleAction rec = this.Actions[i] as RectangleAction;
01084
01085                     GraphicsPath rectanglePath = new GraphicsPath();
01086
01087                     Point pt1 = transformationFunction(Multiply(currMatrix, rec.TopLeft));
01088                     Point pt2 = transformationFunction(Multiply(currMatrix, new Point(rec.TopLeft.X +
rec.Size.Width, rec.TopLeft.Y)));
01089                     Point pt3 = transformationFunction(Multiply(currMatrix, new Point(rec.TopLeft.X +
rec.Size.Width, rec.TopLeft.Y + rec.Size.Height)));
01090                     Point pt4 = transformationFunction(Multiply(currMatrix, new Point(rec.TopLeft.X,
rec.TopLeft.Y + rec.Size.Height)));
01091
01092                     rectanglePath.MoveTo(pt1).LineTo(pt2).LineTo(pt3).LineTo(pt4).Close();
01093
01094                     if (rec.Fill != null)
01095                     {
01096                         destinationGraphics.FillPath(rectanglePath, rec.Fill, rec.Tag);
01097                     }
01098                     else if (rec.Stroke != null)
01099                     {
01100                         destinationGraphics.StrokePath(rectanglePath, rec.Stroke, rec.LineWidth,
rec.LineCap, rec.LineJoin, rec.LineDash, rec.Tag);
01101                     }
01102                 }
01103                 else if (this.Actions[i] is PathAction)
01104                 {
01105                     PathAction pth = this.Actions[i] as PathAction;
01106
01107                     GraphicsPath newPath = pth.Path.Linearise(linearisationResolution).Transform(pt =>
transformationFunction(Multiply(currMatrix, pt)));
01108
01109                     if (pth.IsClipping)
01110                     {
01111                         destinationGraphics.SetClippingPath(newPath);
01112                     }
01113                     else
01114                     {
```

```
01115                         if (pth.Fill != null)
01116                         {
01117                             destinationGraphics.FillPath(newPath, pth.Fill, pth.Tag);
01118                         }
01119                         else if (pth.Stroke != null)
01120                         {
01121                             destinationGraphics.StrokePath(newPath, pth.Stroke, pth.LineWidth,
      pth.LineCap, pth.LineJoin, pth.LineDash, pth.Tag);
01122                         }
01123                     }
01124                 }
01125                 else if (this.Actions[i] is TextAction)
01126                 {
01127                     TextAction txt = this.Actions[i] as TextAction;
01128 
01129                     GraphicsPath textPath = new GraphicsPath().AddText(txt.Origin, txt.Text, txt.Font,
      txt.TextBaseline).Linearise(linearisationResolution).Transform(pt =>
      transformationFunction(Multiply(currMatrix, pt)));
01130 
01131                     if (txt.Fill != null)
01132                     {
01133                         destinationGraphics.FillPath(textPath, txt.Fill, txt.Tag);
01134                     }
01135                     else if (txt.Stroke != null)
01136                     {
01137                         destinationGraphics.StrokePath(textPath, txt.Stroke, txt.LineWidth,
      txt.LineCap, txt.LineJoin, txt.LineDash, txt.Tag);
01138                     }
01139                 }
01140                 else if (this.Actions[i] is TransformAction)
01141                 {
01142                     TransformAction trf = this.Actions[i] as TransformAction;
01143 
01144                     if (trf.Delta != null)
01145                     {
01146                         currMatrix = Multiply(currMatrix, TranslationMatrix(trf.Delta.Value.X,
      trf.Delta.Value.Y));
01147                     }
01148                     else if (trf.Angle != null)
01149                     {
01150                         currMatrix = Multiply(currMatrix, RotationMatrix(trf.Angle.Value));
01151                     }
01152                     else if (trf.Scale != null)
01153                     {
01154                         currMatrix = Multiply(currMatrix, ScaleMatrix(trf.Scale.Value.Width,
      trf.Scale.Value.Height));
01155                     }
01156                     else if (trf.Matrix != null)
01157                     {
01158                         currMatrix = Multiply(currMatrix, trf.Matrix);
01159                     }
01160                 }
01161                 else if (this.Actions[i] is StateAction)
01162                 {
01163                     if (((StateAction)this.Actions[i]).StateActionType ==
      StateAction.StateActionTypes.Save)
01164                     {
01165                         transformMatrix.Push(currMatrix);
01166                     }
01167                     else
01168                     {
01169                         currMatrix = transformMatrix.Pop();
01170                     }
01171                 }
01172                 else if (this.Actions[i] is RasterImageAction)
01173                 {
01174                     RasterImageAction img = this.Actions[i] as RasterImageAction;
01175 
01176                     GraphicsPath rectanglePath = new GraphicsPath();
01177 
01178                     Point pt1 = transformationFunction(Multiply(currMatrix, new
      Point(img.DestinationX, img.DestinationY)));
01179                     Point pt2 = transformationFunction(Multiply(currMatrix, new Point(img.DestinationX
      + img.DestinationWidth, img.DestinationY)));
01180                     Point pt3 = transformationFunction(Multiply(currMatrix, new Point(img.DestinationX
      + img.DestinationWidth, img.DestinationY + img.DestinationHeight)));
01181                     Point pt4 = transformationFunction(Multiply(currMatrix, new
      Point(img.DestinationX, img.DestinationY + img.DestinationHeight)));
01182 
01183                     rectanglePath.MoveTo(pt1).LineTo(pt2).LineTo(pt3).LineTo(pt4).Close();
01184 
01185                     destinationGraphics.FillPath(rectanglePath, Colour.FromRgb(220, 220, 220),
      img.Tag);
01186                 }
01187             }
01188 
01189             return destinationGraphics;
```

```
01190         }
01191
01192         internal static GraphicsPath ReduceMaximumLength(GraphicsPath path, double maxLength)
01193         {
01194             GraphicsPath shortLinearisedPath = new GraphicsPath();
01195
01196             // Square of the max length - so that we avoid the square roots.
01197             double maxLengthSq = maxLength * maxLength;
01198
01199             Point currPoint = new Point();
01200             Point startFigurePoint = new Point();
01201
01202             foreach (Segment seg in path.Segments)
01203             {
01204                 if (seg is MoveSegment mov)
01205                 {
01206                     shortLinearisedPath.MoveTo(mov.Point);
01207                     startFigurePoint = mov.Point;
01208                     currPoint = mov.Point;
01209                 }
01210                 else if (seg is LineSegment line)
01211                 {
01212                     double lengthSq = (line.Point.X - currPoint.X) * (line.Point.X - currPoint.X) +
01213                                       (line.Point.Y - currPoint.Y) * (line.Point.Y - currPoint.Y);
01214
01215                     if (lengthSq < maxLengthSq)
01216                     {
01217                         shortLinearisedPath.LineTo(line.Point);
01218                     }
01219                     else
01220                     {
01221                         int segmentCount = (int)Math.Ceiling(Math.Sqrt(lengthSq / maxLengthSq));
01222
01223                         for (int j = 0; j < segmentCount - 1; j++)
01224                         {
01225                             Point endPoint = new Point(currPoint.X + (line.Point.X - currPoint.X) * (j
      + 1) / segmentCount,
01226                                                        currPoint.Y + (line.Point.Y - currPoint.Y) * (j
      + 1) / segmentCount);
01227                             shortLinearisedPath.LineTo(endPoint);
01228                         }
01229
01230                         shortLinearisedPath.LineTo(line.Point);
01231                     }
01232                     currPoint = line.Point;
01233                 }
01234                 else if (seg is CloseSegment close)
01235                 {
01236                     double lengthSq = (startFigurePoint.X - currPoint.X) * (startFigurePoint.X -
      currPoint.X) +
01237                                       (startFigurePoint.Y - currPoint.Y) * (startFigurePoint.Y -
      currPoint.Y);
01238
01239                     if (lengthSq < maxLengthSq)
01240                     {
01241                         shortLinearisedPath.Close();
01242                     }
01243                     else
01244                     {
01245                         int segmentCount = (int)Math.Ceiling(Math.Sqrt(lengthSq / maxLengthSq));
01246
01247                         for (int j = 0; j < segmentCount - 1; j++)
01248                         {
01249                             Point endPoint = new Point(currPoint.X + (startFigurePoint.X -
      currPoint.X) * (j + 1) / segmentCount,
01250                                                        currPoint.Y + (startFigurePoint.Y -
      currPoint.Y) * (j + 1) / segmentCount);
01251                             shortLinearisedPath.LineTo(endPoint);
01252                         }
01253
01254                         shortLinearisedPath.Close();
01255                     }
01256                     currPoint = startFigurePoint;
01257                 }
01258             }
01259
01260             return shortLinearisedPath;
01261         }
01262
01263 /// <summary>
01264 /// Creates a new <see cref="Graphics"/> object in which all the graphics actions have been
      transformed using an arbitrary transformation function.  Raster images are replaced by grey
      rectangles.
01265 /// </summary>
01266 /// <param name="transformationFunction">An arbitrary transformation function.</param>
01267 /// <param name="linearisationResolution">The resolution that will be used to linearise curve
      segments.</param>
```

```
01268 /// <param name="maxSegmentLength">The maximum length of line segments.</param>
01269 /// <returns>A new <see cref="Graphics"/> object in which all graphics actions have been linearised
      and transformed using the <paramref name="transformationFunction"/>.</returns>
01270         [Pure]
01271         public Graphics Transform(Func<Point, Point> transformationFunction, double
      linearisationResolution, double maxSegmentLength)
01272         {
01273             Graphics destinationGraphics = new Graphics();
01274 
01275             Stack<double[,]> transformMatrix = new Stack<double[,]>();
01276             double[,] currMatrix = new double[3, 3] { { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0, 1 } };
01277 
01278             for (int i = 0; i < this.Actions.Count; i++)
01279             {
01280                 if (this.Actions[i] is RectangleAction)
01281                 {
01282                     RectangleAction rec = this.Actions[i] as RectangleAction;
01283 
01284                     GraphicsPath rectanglePath = new GraphicsPath();
01285 
01286                     Point pt1 = rec.TopLeft;
01287                     Point pt2 = new Point(rec.TopLeft.X + rec.Size.Width, rec.TopLeft.Y);
01288                     Point pt3 = new Point(rec.TopLeft.X + rec.Size.Width, rec.TopLeft.Y +
      rec.Size.Height);
01289                     Point pt4 = new Point(rec.TopLeft.X, rec.TopLeft.Y + rec.Size.Height);
01290 
01291                     rectanglePath.MoveTo(pt1).LineTo(pt2).LineTo(pt3).LineTo(pt4).Close();
01292 
01293                     rectanglePath = ReduceMaximumLength(rectanglePath, maxSegmentLength).Transform(pt
      => transformationFunction(Multiply(currMatrix, pt)));
01294 
01295                     if (rec.Fill != null)
01296                     {
01297                         destinationGraphics.FillPath(rectanglePath, rec.Fill, rec.Tag);
01298                     }
01299                     else if (rec.Stroke != null)
01300                     {
01301                         destinationGraphics.StrokePath(rectanglePath, rec.Stroke, rec.LineWidth,
      rec.LineCap, rec.LineJoin, rec.LineDash, rec.Tag);
01302                     }
01303                 }
01304                 else if (this.Actions[i] is PathAction)
01305                 {
01306                     PathAction pth = this.Actions[i] as PathAction;
01307 
01308                     GraphicsPath newPath =
      ReduceMaximumLength(pth.Path.Linearise(linearisationResolution), maxSegmentLength).Transform(pt =>
      transformationFunction(Multiply(currMatrix, pt)));
01309 
01310                     if (pth.IsClipping)
01311                     {
01312                         destinationGraphics.SetClippingPath(newPath);
01313                     }
01314                     else
01315                     {
01316                         if (pth.Fill != null)
01317                         {
01318                             destinationGraphics.FillPath(newPath, pth.Fill, pth.Tag);
01319                         }
01320                         else if (pth.Stroke != null)
01321                         {
01322                             destinationGraphics.StrokePath(newPath, pth.Stroke, pth.LineWidth,
      pth.LineCap, pth.LineJoin, pth.LineDash, pth.Tag);
01323                         }
01324                     }
01325                 }
01326                 else if (this.Actions[i] is TextAction)
01327                 {
01328                     TextAction txt = this.Actions[i] as TextAction;
01329 
01330                     GraphicsPath textPath = ReduceMaximumLength(new GraphicsPath().AddText(txt.Origin,
      txt.Text, txt.Font, txt.TextBaseline).Linearise(linearisationResolution),
      maxSegmentLength).Transform(pt => transformationFunction(Multiply(currMatrix, pt)));
01331 
01332                     if (txt.Fill != null)
01333                     {
01334                         destinationGraphics.FillPath(textPath, txt.Fill, txt.Tag);
01335                     }
01336                     else if (txt.Stroke != null)
01337                     {
01338                         destinationGraphics.StrokePath(textPath, txt.Stroke, txt.LineWidth,
      txt.LineCap, txt.LineJoin, txt.LineDash, txt.Tag);
01339                     }
01340                 }
01341                 else if (this.Actions[i] is TransformAction)
01342                 {
01343                     TransformAction trf = this.Actions[i] as TransformAction;
```

```
01344
01345                     if (trf.Delta != null)
01346                     {
01347                         currMatrix = Multiply(currMatrix, TranslationMatrix(trf.Delta.Value.X,
      trf.Delta.Value.Y));
01348                     }
01349                     else if (trf.Angle != null)
01350                     {
01351                         currMatrix = Multiply(currMatrix, RotationMatrix(trf.Angle.Value));
01352                     }
01353                     else if (trf.Scale != null)
01354                     {
01355                         currMatrix = Multiply(currMatrix, ScaleMatrix(trf.Scale.Value.Width,
      trf.Scale.Value.Height));
01356                     }
01357                     else if (trf.Matrix != null)
01358                     {
01359                         currMatrix = Multiply(currMatrix, trf.Matrix);
01360                     }
01361                 }
01362                 else if (this.Actions[i] is StateAction)
01363                 {
01364                     if (((StateAction)this.Actions[i]).StateActionType ==
      StateAction.StateActionTypes.Save)
01365                     {
01366                         transformMatrix.Push(currMatrix);
01367                     }
01368                     else
01369                     {
01370                         currMatrix = transformMatrix.Pop();
01371                     }
01372                 }
01373                 else if (this.Actions[i] is RasterImageAction)
01374                 {
01375                     RasterImageAction img = this.Actions[i] as RasterImageAction;
01376
01377                     GraphicsPath rectanglePath = new GraphicsPath();
01378
01379                     Point pt1 = new Point(img.DestinationX, img.DestinationY);
01380                     Point pt2 = new Point(img.DestinationX + img.DestinationWidth, img.DestinationY);
01381                     Point pt3 = new Point(img.DestinationX + img.DestinationWidth, img.DestinationY +
      img.DestinationHeight);
01382                     Point pt4 = new Point(img.DestinationX, img.DestinationY + img.DestinationHeight);
01383
01384                     rectanglePath.MoveTo(pt1).LineTo(pt2).LineTo(pt3).LineTo(pt4).Close();
01385
01386                     rectanglePath = ReduceMaximumLength(rectanglePath, maxSegmentLength).Transform(pt
      => transformationFunction(Multiply(currMatrix, pt)));
01387
01388                     destinationGraphics.FillPath(rectanglePath, Colour.FromRgb(220, 220, 220),
      img.Tag);
01389                 }
01390             }
01391
01392             return destinationGraphics;
01393         }
01394
01395 /// <summary>
01396 /// Creates a new <see cref="Graphics"/> object by linearising all of the elements of the current
      instance, i.e.  replacing curve segments with series of line segments that approximate them.  Raster
      images are left unchanged.
01397 /// </summary>
01398 /// <param name="resolution">The resolution that will be used to linearise curve segments.</param>
01399 /// <returns>A new <see cref="Graphics"/> object containing the linearised elements.</returns>
01400         [Pure]
01401         public Graphics Linearise(double resolution)
01402         {
01403             Graphics destinationGraphics = new Graphics();
01404
01405             for (int i = 0; i < this.Actions.Count; i++)
01406             {
01407                 if (this.Actions[i] is RectangleAction || this.Actions[i] is TransformAction ||
      this.Actions[i] is StateAction || this.Actions[i] is RasterImageAction)
01408                 {
01409                     destinationGraphics.Actions.Add(this.Actions[i]);
01410                 }
01411                 else if (this.Actions[i] is PathAction)
01412                 {
01413                     PathAction pth = this.Actions[i] as PathAction;
01414
01415                     GraphicsPath newPath = pth.Path.Linearise(resolution);
01416
01417                     if (pth.IsClipping)
01418                     {
01419                         destinationGraphics.SetClippingPath(newPath);
01420                     }
01421                     else
```

```
01422                     {
01423                         if (pth.Fill != null)
01424                         {
01425                             destinationGraphics.FillPath(newPath, pth.Fill, pth.Tag);
01426                         }
01427                         else if (pth.Stroke != null)
01428                         {
01429                             destinationGraphics.StrokePath(newPath, pth.Stroke, pth.LineWidth,
      pth.LineCap, pth.LineJoin, pth.LineDash, pth.Tag);
01430                         }
01431                     }
01432                 }
01433                 else if (this.Actions[i] is TextAction)
01434                 {
01435                     TextAction txt = this.Actions[i] as TextAction;
01436 
01437                     GraphicsPath textPath = new GraphicsPath().AddText(txt.Origin, txt.Text, txt.Font,
      txt.TextBaseline).Linearise(resolution);
01438 
01439                     if (txt.Fill != null)
01440                     {
01441                         destinationGraphics.FillPath(textPath, txt.Fill, txt.Tag);
01442                     }
01443                     else if (txt.Stroke != null)
01444                     {
01445                         destinationGraphics.StrokePath(textPath, txt.Stroke, txt.LineWidth,
      txt.LineCap, txt.LineJoin, txt.LineDash, txt.Tag);
01446                     }
01447                 }
01448             }
01449 
01450             return destinationGraphics;
01451         }
01452 
01453 /// <summary>
01454 /// Computes the rectangular bounds of the region affected by the drawing operations performed on the
      <see cref="Graphics"/> object.
01455 /// </summary>
01456 /// <returns>The smallest rectangle that contains all the elements drawn on the <see
      cref="Graphics"/>.</returns>
01457         [Pure]
01458         public Rectangle GetBounds()
01459         {
01460             Rectangle tbr = Rectangle.NaN;
01461             bool initialised = false;
01462 
01463             Stack<Rectangle> clippingPaths = new Stack<Rectangle>();
01464             Rectangle currClippingPath = Rectangle.NaN;
01465 
01466             Stack<double[,]> transformMatrix = new Stack<double[,]>();
01467             double[,] currMatrix = new double[3, 3] { { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0, 1 } };
01468 
01469             for (int i = 0; i < this.Actions.Count; i++)
01470             {
01471                 if (this.Actions[i] is IPrintableAction act)
01472                 {
01473                     Rectangle bounds = act.GetBounds();
01474 
01475                     double lineThickness = double.IsNaN(act.LineWidth) ?  0 :  act.LineWidth;
01476 
01477                     Point pt1 = Multiply(currMatrix, new Point(bounds.Location.X - lineThickness *
      0.5, bounds.Location.Y - lineThickness * 0.5));
01478                     Point pt2 = Multiply(currMatrix, new Point(bounds.Location.X + bounds.Size.Width +
      lineThickness * 0.5, bounds.Location.Y - lineThickness * 0.5));
01479                     Point pt3 = Multiply(currMatrix, new Point(bounds.Location.X + bounds.Size.Width +
      lineThickness * 0.5, bounds.Location.Y + bounds.Size.Height + lineThickness * 0.5));
01480                     Point pt4 = Multiply(currMatrix, new Point(bounds.Location.X - lineThickness *
      0.5, bounds.Location.Y + bounds.Size.Height + lineThickness * 0.5));
01481 
01482                     bounds = Point.Bounds(pt1, pt2, pt3, pt4);
01483 
01484                     if (act is PathAction pth && pth.IsClipping)
01485                     {
01486                         currClippingPath = bounds;
01487                     }
01488                     else
01489                     {
01490                         if (!currClippingPath.IsNaN())
01491                         {
01492                             bounds = Rectangle.Intersection(bounds, currClippingPath);
01493                         }
01494 
01495                         if (!double.IsNaN(bounds.Location.X) && !double.IsNaN(bounds.Location.Y) &&
      !double.IsNaN(bounds.Size.Width) && !double.IsNaN(bounds.Size.Height))
01496                         {
01497                             if (!initialised)
01498                             {
```

```
01499                                   tbr = bounds;
01500                                   initialised = true;
01501                               }
01502                               else
01503                               {
01504                                   tbr = Rectangle.Union(tbr, bounds);
01505                               }
01506                           }
01507                       }
01508                   }
01509                   else if (this.Actions[i] is TransformAction)
01510                   {
01511                       TransformAction trf = this.Actions[i] as TransformAction;
01512
01513                       if (trf.Delta != null)
01514                       {
01515                           currMatrix = Multiply(currMatrix, TranslationMatrix(trf.Delta.Value.X,
      trf.Delta.Value.Y));
01516                       }
01517                       else if (trf.Angle != null)
01518                       {
01519                           currMatrix = Multiply(currMatrix, RotationMatrix(trf.Angle.Value));
01520                       }
01521                       else if (trf.Scale != null)
01522                       {
01523                           currMatrix = Multiply(currMatrix, ScaleMatrix(trf.Scale.Value.Width,
      trf.Scale.Value.Height));
01524                       }
01525                       else if (trf.Matrix != null)
01526                       {
01527                           currMatrix = Multiply(currMatrix, trf.Matrix);
01528                       }
01529                   }
01530                   else if (this.Actions[i] is StateAction)
01531                   {
01532                       if (((StateAction)this.Actions[i]).StateActionType ==
      StateAction.StateActionTypes.Save)
01533                       {
01534                           transformMatrix.Push(currMatrix);
01535                           clippingPaths.Push(currClippingPath);
01536                       }
01537                       else
01538                       {
01539                           currMatrix = transformMatrix.Pop();
01540                           currClippingPath = clippingPaths.Pop();
01541                       }
01542                   }
01543               }
01544
01545               return tbr;
01546           }
01547
01548 /// <summary>
01549 /// A method that is used to rasterise a region of a <see cref="Graphics"/> object.  Set this to <see
      langword="null"/> if you wish to use the default
01550 /// rasterisation methods (implemented by either VectSharp.Raster, or VectSharp.Raster.ImageSharp).
      You will have to provide your own implementation
01551 /// of this method if neither VectSharp.Raster nor VectSharp.Raster.ImageSharp are referenced by your
      project.  The first argument of this method is
01552 /// the <see cref="Graphics"/> to be rasterised, the second is a <see cref="Rectangle"/> representing
      the region to rasterise, the third is a
01553 /// <see cref="double"/> representing the scale, and the third is a boolean value indicating whether
      the resulting <see cref="RasterImage"/> should
01554 /// be interpolated.
01555 /// </summary>
01556           public static Func<Graphics, Rectangle, double, bool, RasterImage> RasterisationMethod = null;
01557
01558 /// <summary>
01559 /// Tries to rasterise specified region of this <see cref="Graphics"/> object using the default
      rasterisation method.
01560 /// </summary>
01561 /// <param name="region">The region of the <see cref="Graphics"/> to rasterise.</param>
01562 /// <param name="scale">The scale at which the image is rasterised.</param>
01563 /// <param name="interpolate">Determines whether the resulting <see cref="RasterImage"/> should be
      interpolated or not.</param>
01564 /// <param name="output">When this method returns, this will contain the rasterised image (or <see
      langword="null"/> if the image could not be rasterised.</param>
01565 /// <returns><see langword="true"/> if the image could be rasterised; <see langword="false"/> if it
      could not be rasterised.</returns>
01566           public bool TryRasterise(Rectangle region, double scale, bool interpolate, out RasterImage
      output)
01567           {
01568               if (RasterisationMethod != null)
01569               {
01570                   output = RasterisationMethod(this, region, scale, interpolate);
01571                   return true;
01572               }
```

```
01573             else
01574             {
01575                 System.Reflection.Assembly raster;
01576 
01577                 try
01578                 {
01579                     raster = System.Reflection.Assembly.Load("VectSharp.Raster");
01580                     if (raster != null)
01581                     {
01582                         System.Reflection.MethodInfo rasteriser =
      raster.GetType("VectSharp.Raster.Raster").GetMethod("Rasterise", System.Reflection.BindingFlags.Public
      | System.Reflection.BindingFlags.Static);
01583                         output = (RasterImage)rasteriser.Invoke(null, new object[] { this, region,
      scale, interpolate });
01584                         return true;
01585                     }
01586                 }
01587                 catch { }
01588 
01589                 try
01590                 {
01591                     raster = System.Reflection.Assembly.Load("VectSharp.Raster.ImageSharp");
01592                     if (raster != null)
01593                     {
01594                         System.Reflection.MethodInfo rasteriser =
      raster.GetType("VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter").GetMethod("Rasterise",
      System.Reflection.BindingFlags.Public | System.Reflection.BindingFlags.Static);
01595                         output = (RasterImage)rasteriser.Invoke(null, new object[] { this, region,
      scale, interpolate });
01596                         return true;
01597                     }
01598                 }
01599                 catch { }
01600 
01601 
01602                 output = null;
01603                 return false;
01604             }
01605         }
01606 
01607 /// <summary>
01608 /// Removes graphics actions that fall completely outside of the specified <paramref name="region"/>.
01609 /// </summary>
01610 /// <param name="region">The area to preserve.</param>
01611         public void Crop(Rectangle region)
01612         {
01613             List<int> itemsToRemove = new List<int>();
01614 
01615             Stack<Rectangle> clippingPaths = new Stack<Rectangle>();
01616             Rectangle currClippingPath = Rectangle.NaN;
01617 
01618             Stack<double[,]> transformMatrix = new Stack<double[,]>();
01619             double[,] currMatrix = new double[3, 3] { { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0, 1 } };
01620 
01621             for (int i = 0; i < this.Actions.Count; i++)
01622             {
01623                 if (this.Actions[i] is IPrintableAction act)
01624                 {
01625                     Rectangle bounds = act.GetBounds();
01626 
01627                     double lineThickness = double.IsNaN(act.LineWidth) ?  0 :  act.LineWidth;
01628 
01629                     Point pt1 = Multiply(currMatrix, new Point(bounds.Location.X - lineThickness *
      0.5, bounds.Location.Y - lineThickness * 0.5));
01630                     Point pt2 = Multiply(currMatrix, new Point(bounds.Location.X + bounds.Size.Width +
      lineThickness * 0.5, bounds.Location.Y - lineThickness * 0.5));
01631                     Point pt3 = Multiply(currMatrix, new Point(bounds.Location.X + bounds.Size.Width +
      lineThickness * 0.5, bounds.Location.Y + bounds.Size.Height + lineThickness * 0.5));
01632                     Point pt4 = Multiply(currMatrix, new Point(bounds.Location.X - lineThickness *
      0.5, bounds.Location.Y + bounds.Size.Height + lineThickness * 0.5));
01633 
01634                     bounds = Point.Bounds(pt1, pt2, pt3, pt4);
01635 
01636                     if (act is PathAction pth && pth.IsClipping)
01637                     {
01638                         currClippingPath = bounds;
01639                     }
01640                     else
01641                     {
01642                         if (!currClippingPath.IsNaN())
01643                         {
01644                             bounds = Rectangle.Intersection(bounds, currClippingPath);
01645                         }
01646 
01647                         if (double.IsNaN(bounds.Location.X) || double.IsNaN(bounds.Location.Y) ||
      double.IsNaN(bounds.Size.Width) || double.IsNaN(bounds.Size.Height) || Rectangle.Intersection(bounds,
      region).IsNaN())
```

```
01648                           {
01649                               itemsToRemove.Add(i);
01650                           }
01651                       }
01652                   }
01653                   else if (this.Actions[i] is TransformAction)
01654                   {
01655                       TransformAction trf = this.Actions[i] as TransformAction;
01656 
01657                       if (trf.Delta != null)
01658                       {
01659                           currMatrix = Multiply(currMatrix, TranslationMatrix(trf.Delta.Value.X,
      trf.Delta.Value.Y));
01660                       }
01661                       else if (trf.Angle != null)
01662                       {
01663                           currMatrix = Multiply(currMatrix, RotationMatrix(trf.Angle.Value));
01664                       }
01665                       else if (trf.Scale != null)
01666                       {
01667                           currMatrix = Multiply(currMatrix, ScaleMatrix(trf.Scale.Value.Width,
      trf.Scale.Value.Height));
01668                       }
01669                       else if (trf.Matrix != null)
01670                       {
01671                           currMatrix = Multiply(currMatrix, trf.Matrix);
01672                       }
01673                   }
01674                   else if (this.Actions[i] is StateAction)
01675                   {
01676                       if (((StateAction)this.Actions[i]).StateActionType ==
      StateAction.StateActionTypes.Save)
01677                       {
01678                           transformMatrix.Push(currMatrix);
01679                           clippingPaths.Push(currClippingPath);
01680                       }
01681                       else
01682                       {
01683                           currMatrix = transformMatrix.Pop();
01684                           currClippingPath = clippingPaths.Pop();
01685                       }
01686                   }
01687               }
01688 
01689               for (int i = itemsToRemove.Count - 1; i >= 0; i--)
01690               {
01691                   this.Actions.RemoveAt(itemsToRemove[i]);
01692               }
01693           }
01694 
01695 /// <summary>
01696 /// Removes graphics actions that fall completely outside of the specified region.
01697 /// </summary>
01698 /// <param name="topLeft">The top-left corner of the area to preserve.</param>
01699 /// <param name="size">The size of the area to preserve.</param>
01700           public void Crop(Point topLeft, Size size)
01701           {
01702               Rectangle region = new Rectangle(topLeft, size);
01703               this.Crop(region);
01704           }
01705 
01706 /// <summary>
01707 /// Gets all the tags that have been defined in the <see cref="Graphics"/>.
01708 /// </summary>
01709 /// <returns>An <see cref="IEnumerable{T}"/> of <see cref="string"/>s that, when enumerated, returns
      all the tags that have been defined in the <see cref="Graphics"/>.</returns>
01710           public IEnumerable<string> GetTags()
01711           {
01712               foreach (IGraphicsAction action in this.Actions)
01713               {
01714                   if (!string.IsNullOrEmpty(action.Tag))
01715                   {
01716                       yield return action.Tag;
01717                   }
01718               }
01719           }
01720       }
01721 }
```

8.84 Graphics.Text.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
```

```
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020
00021 namespace VectSharp
00022 {
00023     public partial class Graphics
00024     {
00025 /// <summary>
00026 /// Fill a text string.
00027 /// </summary>
00028 /// <param name="origin">The text origin.  See <paramref name="textBaseline"/>.</param>
00029 /// <param name="text">The string to draw.</param>
00030 /// <param name="font">The font with which to draw the text.</param>
00031 /// <param name="fillColour">The <see cref="Brush"/> to use to fill the text.</param>
00032 /// <param name="textBaseline">The text baseline (determines what the vertical component of <paramref
      name="origin"/> represents).</param>
00033 /// <param name="tag">A tag to identify the filled text.</param>
00034         public void FillText(Point origin, string text, Font font, Brush fillColour, TextBaselines
      textBaseline = TextBaselines.Top, string tag = null)
00035         {
00036             if (!string.IsNullOrEmpty(text))
00037             {
00038                 Actions.Add(new TextAction(origin, text, font, textBaseline, fillColour, null, 0,
      LineCaps.Butt, LineJoins.Miter, LineDash.SolidLine, tag));
00039
00040                 if (font.Underline != null)
00041                 {
00042                     FillTextUnderline(origin, text, font, fillColour, textBaseline,
      (!string.IsNullOrEmpty(tag) && UseUniqueTags) ?  (tag + "@underline") :  tag);
00043                 }
00044             }
00045         }
00046
00047 /// <summary>
00048 /// Fill a text string.
00049 /// </summary>
00050 /// <param name="origin">The text origin.  See <paramref name="textBaseline"/>.</param>
00051 /// <param name="text">The string to draw.</param>
00052 /// <param name="font">The font with which to draw the text.</param>
00053 /// <param name="fillColour">The <see cref="Brush"/> to use to fill the text.</param>
00054 /// <param name="spacing">Spacing parameters to use for the text.</param>
00055 /// <param name="textBaseline">The text baseline (determines what the vertical component of <paramref
      name="origin"/> represents).</param>
00056 /// <param name="tag">A tag to identify the filled text.</param>
00057         public void FillText(Point origin, string text, Font font, Brush fillColour, TextSpacing
      spacing, TextBaselines textBaseline = TextBaselines.Top, string tag = null)
00058         {
00059             if (spacing == TextSpacing.Default)
00060             {
00061                 FillText(origin, text, font, fillColour, textBaseline, tag);
00062             }
00063             else
00064             {
00065                 FillText(origin, new FormattedText[] { new FormattedText(text, font, spacing:
      spacing) }, fillColour, textBaseline, tag);
00066             }
00067         }
00068
00069 /// <summary>
00070 /// Fill a text string.
00071 /// </summary>
00072 /// <param name="originX">The horizontal coordinate of the text origin.</param>
00073 /// <param name="originY">The vertical coordinate of the text origin.  See <paramref
      name="textBaseline"/>.</param>
00074 /// <param name="text">The string to draw.</param>
00075 /// <param name="font">The font with which to draw the text.</param>
00076 /// <param name="fillColour">The <see cref="Brush"/> to use to fill the text.</param>
00077 /// <param name="textBaseline">The text baseline (determines what <paramref name="originY"/>
      represents).</param>
00078 /// <param name="tag">A tag to identify the filled text.</param>
00079         public void FillText(double originX, double originY, string text, Font font, Brush fillColour,
      TextBaselines textBaseline = TextBaselines.Top, string tag = null)
```

```
00080         {
00081             if (!string.IsNullOrEmpty(text))
00082             {
00083                 Actions.Add(new TextAction(new Point(originX, originY), text, font, textBaseline,
      fillColour, null, 0, LineCaps.Butt, LineJoins.Miter, LineDash.SolidLine, tag));
00084 
00085                 if (font.Underline != null)
00086                 {
00087                     FillTextUnderline(originX, originY, text, font, fillColour, textBaseline,
      (!string.IsNullOrEmpty(tag) && UseUniqueTags) ?  (tag + "@underline") :  tag);
00088                 }
00089             }
00090         }
00091 
00092 /// <summary>
00093 /// Fill a text string.
00094 /// </summary>
00095 /// <param name="originX">The horizontal coordinate of the text origin.</param>
00096 /// <param name="originY">The vertical coordinate of the text origin.  See <paramref
      name="textBaseline"/>.</param>
00097 /// <param name="text">The string to draw.</param>
00098 /// <param name="font">The font with which to draw the text.</param>
00099 /// <param name="fillColour">The <see cref="Brush"/> to use to fill the text.</param>
00100 /// <param name="spacing">Spacing parameters to use for the text.</param>
00101 /// <param name="textBaseline">The text baseline (determines what <paramref name="originY"/>
      represents).</param>
00102 /// <param name="tag">A tag to identify the filled text.</param>
00103         public void FillText(double originX, double originY, string text, Font font, Brush fillColour,
      TextSpacing spacing, TextBaselines textBaseline = TextBaselines.Top, string tag = null)
00104         {
00105             if (spacing == TextSpacing.Default)
00106             {
00107                 FillText(originX, originY, text, font, fillColour, textBaseline, tag);
00108             }
00109             else
00110             {
00111                 FillText(originX, originY, new FormattedText[] { new FormattedText(text, font,
      spacing:  spacing) }, fillColour, textBaseline, tag);
00112             }
00113         }
00114 
00115 /// <summary>
00116 /// Stroke a text string.
00117 /// </summary>
00118 /// <param name="origin">The text origin.  See <paramref name="textBaseline"/>.</param>
00119 /// <param name="text">The string to draw.</param>
00120 /// <param name="font">The font with which to draw the text.</param>
00121 /// <param name="strokeColour">The <see cref="Brush"/> with which to stroke the text.</param>
00122 /// <param name="lineWidth">The width of the line with which the text is stroked.</param>
00123 /// <param name="lineCap">The line cap to use to stroke the text.</param>
00124 /// <param name="lineJoin">The line join to use to stroke the text.</param>
00125 /// <param name="lineDash">The line dash to use to stroke the text.</param>
00126 /// <param name="textBaseline">The text baseline (determines what the vertical component of <paramref
      name="origin"/> represents).</param>
00127 /// <param name="tag">A tag to identify the stroked text.</param>
00128         public void StrokeText(Point origin, string text, Font font, Brush strokeColour, TextBaselines
      textBaseline = TextBaselines.Top, double lineWidth = 1, LineCaps lineCap = LineCaps.Butt, LineJoins
      lineJoin = LineJoins.Miter, LineDash?  lineDash = null, string tag = null)
00129         {
00130             if (!string.IsNullOrEmpty(text))
00131             {
00132                 Actions.Add(new TextAction(origin, text, font, textBaseline, null, strokeColour,
      lineWidth, lineCap, lineJoin, lineDash ??  LineDash.SolidLine, tag));
00133 
00134                 if (font.Underline != null)
00135                 {
00136                     StrokeTextUnderline(origin, text, font, strokeColour, textBaseline, lineWidth,
      lineCap, lineJoin, lineDash, (!string.IsNullOrEmpty(tag) && UseUniqueTags) ?  (tag + "@underline") :
      tag);
00137                 }
00138             }
00139         }
00140 
00141 /// <summary>
00142 /// Stroke a text string.
00143 /// </summary>
00144 /// <param name="origin">The text origin.  See <paramref name="textBaseline"/>.</param>
00145 /// <param name="text">The string to draw.</param>
00146 /// <param name="font">The font with which to draw the text.</param>
00147 /// <param name="strokeColour">The <see cref="Brush"/> with which to stroke the text.</param>
00148 /// <param name="spacing">Spacing parameters to use for the text.</param>
00149 /// <param name="lineWidth">The width of the line with which the text is stroked.</param>
00150 /// <param name="lineCap">The line cap to use to stroke the text.</param>
00151 /// <param name="lineJoin">The line join to use to stroke the text.</param>
00152 /// <param name="lineDash">The line dash to use to stroke the text.</param>
00153 /// <param name="textBaseline">The text baseline (determines what the vertical component of <paramref
      name="origin"/> represents).</param>
```

```
00154 /// <param name="tag">A tag to identify the stroked text.</param>
00155         public void StrokeText(Point origin, string text, Font font, Brush strokeColour, TextSpacing
      spacing, TextBaselines textBaseline = TextBaselines.Top, double lineWidth = 1, LineCaps lineCap =
      LineCaps.Butt, LineJoins lineJoin = LineJoins.Miter, LineDash?  lineDash = null, string tag = null)
00156         {
00157             if (spacing == TextSpacing.Default)
00158             {
00159                 StrokeText(origin, text, font, strokeColour, textBaseline, lineWidth, lineCap,
      lineJoin, lineDash, tag);
00160             }
00161             else
00162             {
00163                 StrokeText(origin, new FormattedText[] { new FormattedText(text, font, spacing:
      spacing) }, strokeColour, textBaseline, lineWidth, lineCap, lineJoin, lineDash, tag);
00164             }
00165         }
00166 
00167 /// <summary>
00168 /// Stroke a text string.
00169 /// </summary>
00170 /// <param name="originX">The horizontal coordinate of the text origin.</param>
00171 /// <param name="originY">The vertical coordinate of the text origin.  See <paramref
      name="textBaseline"/>.</param>
00172 /// <param name="text">The string to draw.</param>
00173 /// <param name="font">The font with which to draw the text.</param>
00174 /// <param name="strokeColour">The <see cref="Brush"/> with which to stroke the text.</param>
00175 /// <param name="lineWidth">The width of the line with which the text is stroked.</param>
00176 /// <param name="lineCap">The line cap to use to stroke the text.</param>
00177 /// <param name="lineJoin">The line join to use to stroke the text.</param>
00178 /// <param name="lineDash">The line dash to use to stroke the text.</param>
00179 /// <param name="textBaseline">The text baseline (determines what <paramref name="originY"/>
      represents).</param>
00180 /// <param name="tag">A tag to identify the stroked text.</param>
00181         public void StrokeText(double originX, double originY, string text, Font font, Brush
      strokeColour, TextBaselines textBaseline = TextBaselines.Top, double lineWidth = 1, LineCaps lineCap =
      LineCaps.Butt, LineJoins lineJoin = LineJoins.Miter, LineDash?  lineDash = null, string tag = null)
00182         {
00183             if (!string.IsNullOrEmpty(text))
00184             {
00185                 Actions.Add(new TextAction(new Point(originX, originY), text, font, textBaseline,
      null, strokeColour, lineWidth, lineCap, lineJoin, lineDash ??  LineDash.SolidLine, tag));
00186 
00187                 if (font.Underline != null)
00188                 {
00189                     StrokeTextUnderline(originX, originY, text, font, strokeColour, textBaseline,
      lineWidth, lineCap, lineJoin, lineDash, (!string.IsNullOrEmpty(tag) && UseUniqueTags) ?  (tag +
      "@underline") :  tag);
00190                 }
00191             }
00192         }
00193 
00194 /// <summary>
00195 /// Stroke a text string.
00196 /// </summary>
00197 /// <param name="originX">The horizontal coordinate of the text origin.</param>
00198 /// <param name="originY">The vertical coordinate of the text origin.  See <paramref
      name="textBaseline"/>.</param>
00199 /// <param name="text">The string to draw.</param>
00200 /// <param name="font">The font with which to draw the text.</param>
00201 /// <param name="strokeColour">The <see cref="Brush"/> with which to stroke the text.</param>
00202 /// <param name="spacing">Spacing parameters to use for the text.</param>
00203 /// <param name="lineWidth">The width of the line with which the text is stroked.</param>
00204 /// <param name="lineCap">The line cap to use to stroke the text.</param>
00205 /// <param name="lineJoin">The line join to use to stroke the text.</param>
00206 /// <param name="lineDash">The line dash to use to stroke the text.</param>
00207 /// <param name="textBaseline">The text baseline (determines what <paramref name="originY"/>
      represents).</param>
00208 /// <param name="tag">A tag to identify the stroked text.</param>
00209         public void StrokeText(double originX, double originY, string text, Font font, Brush
      strokeColour, TextSpacing spacing, TextBaselines textBaseline = TextBaselines.Top, double lineWidth =
      1, LineCaps lineCap = LineCaps.Butt, LineJoins lineJoin = LineJoins.Miter, LineDash?  lineDash = null,
      string tag = null)
00210         {
00211             if (spacing == TextSpacing.Default)
00212             {
00213                 StrokeText(originX, originY, text, font, strokeColour, textBaseline, lineWidth,
      lineCap, lineJoin, lineDash, tag);
00214             }
00215             else
00216             {
00217                 StrokeText(originX, originY, new FormattedText[] { new FormattedText(text, font,
      spacing:  spacing) }, strokeColour, textBaseline, lineWidth, lineCap, lineJoin, lineDash, tag);
00218             }
00219         }
00220 
00221 /// <summary>
00222 /// Fill a text string along a <see cref="GraphicsPath"/>.
```

```
00223 /// </summary>
00224 /// <param name="path">The <see cref="GraphicsPath"/> along which the text will flow.</param>
00225 /// <param name="text">The string to draw.</param>
00226 /// <param name="font">The font with which to draw the text.</param>
00227 /// <param name="fillColour">The <see cref="Brush"/> to use to fill the text.</param>
00228 /// <param name="reference">The (relative) starting point on the path starting from which the text
      should be drawn (0 is the start of the path, 1 is the end of the path).</param>
00229 /// <param name="anchor">The anchor in the text string that will correspond to the point specified by
      the <paramref name="reference"/>.</param>
00230 /// <param name="textBaseline">The text baseline (determines which the position of the text in
      relation to the <paramref name="path"/>.</param>
00231 /// <param name="tag">A tag to identify the filled text.</param>
00232         public void FillTextOnPath(GraphicsPath path, string text, Font font, Brush fillColour, double
      reference = 0, TextAnchors anchor = TextAnchors.Left, TextBaselines textBaseline = TextBaselines.Top,
      string tag = null)
00233         {
00234             if (!string.IsNullOrEmpty(text))
00235             {
00236                 if (font.Underline != null)
00237                 {
00238                     font = new Font(font.FontFamily, font.FontSize, false);
00239                 }
00240 
00241                 double currDelta = 0;
00242                 double pathLength = path.MeasureLength();
00243 
00244                 Font.DetailedFontMetrics fullMetrics = font.MeasureTextAdvanced(text);
00245 
00246                 switch (anchor)
00247                 {
00248                     case TextAnchors.Left:
00249                         break;
00250                     case TextAnchors.Center:
00251                         currDelta = -fullMetrics.Width * 0.5 / pathLength;
00252                         break;
00253                     case TextAnchors.Right:
00254                         currDelta = -fullMetrics.Width / pathLength;
00255                         break;
00256                 }
00257 
00258                 Point currentGlyphPlacementDelta = new Point();
00259                 Point currentGlyphAdvanceDelta = new Point();
00260                 Point nextGlyphPlacementDelta = new Point();
00261                 Point nextGlyphAdvanceDelta = new Point();
00262 
00263                 for (int i = 0; i < text.Length; i++)
00264                 {
00265                     string c = text.Substring(i, 1);
00266 
00267                     if (Font.EnableKerning && i < text.Length - 1)
00268                     {
00269                         currentGlyphPlacementDelta = nextGlyphPlacementDelta;
00270                         currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
00271                         nextGlyphAdvanceDelta = new Point();
00272                         nextGlyphPlacementDelta = new Point();
00273 
00274                         TrueTypeFile.PairKerning kerning =
      font.FontFamily.TrueTypeFile.Get1000EmKerning(text[i], text[i + 1]);
00275 
00276                         if (kerning != null)
00277                         {
00278                             currentGlyphPlacementDelta = new Point(currentGlyphPlacementDelta.X +
      kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y + kerning.Glyph1Placement.Y);
00279                             currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
      kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
00280 
00281                             nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
      kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
00282                             nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
      kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
00283                         }
00284                     }
00285 
00286                     Font.DetailedFontMetrics metrics = font.MeasureTextAdvanced(c);
00287 
00288                     Point origin = path.GetPointAtRelative(reference + currDelta +
      currentGlyphPlacementDelta.X * font.FontSize / 1000);
00289 
00290                     Point tangent = path.GetTangentAtRelative(reference + currDelta +
      currentGlyphPlacementDelta.X * font.FontSize / 1000 + (metrics.Width + metrics.RightSideBearing +
      metrics.LeftSideBearing) / pathLength * 0.5);
00291 
00292                     origin = new Point(origin.X - tangent.Y * currentGlyphPlacementDelta.Y *
      font.FontSize / 1000, origin.Y + tangent.X * currentGlyphPlacementDelta.Y * font.FontSize / 1000);
00293 
00294                     this.Save();
00295 
```

```
00296                     this.Translate(origin);
00297                     this.Rotate(Math.Atan2(tangent.Y, tangent.X));
00298 
00299                     string glyphTag = (!string.IsNullOrEmpty(tag) && UseUniqueTags) ?  (tag + "@" +
      i.ToString()) :  tag;
00300 
00301                     switch (textBaseline)
00302                     {
00303                         case TextBaselines.Top:
00304                             if (i > 0)
00305                             {
00306                                 this.FillText(new Point(metrics.LeftSideBearing, fullMetrics.Top), c,
      font, fillColour, textBaseline:  TextBaselines.Baseline, glyphTag);
00307                             }
00308                             else
00309                             {
00310                                 this.FillText(new Point(0, fullMetrics.Top), c, font, fillColour,
      textBaseline:  TextBaselines.Baseline, glyphTag);
00311                             }
00312                             break;
00313                         case TextBaselines.Baseline:
00314                             if (i > 0)
00315                             {
00316                                 this.FillText(new Point(metrics.LeftSideBearing, 0), c, font,
      fillColour, textBaseline:  TextBaselines.Baseline, glyphTag);
00317                             }
00318                             else
00319                             {
00320                                 this.FillText(new Point(0, 0), c, font, fillColour, textBaseline:
      TextBaselines.Baseline, glyphTag);
00321                             }
00322                             break;
00323                         case TextBaselines.Bottom:
00324                             if (i > 0)
00325                             {
00326                                 this.FillText(new Point(metrics.LeftSideBearing, fullMetrics.Bottom),
      c, font, fillColour, textBaseline:  TextBaselines.Baseline, glyphTag);
00327                             }
00328                             else
00329                             {
00330                                 this.FillText(new Point(0, fullMetrics.Bottom), c, font, fillColour,
      textBaseline:  TextBaselines.Baseline, glyphTag);
00331                             }
00332                             break;
00333                         case TextBaselines.Middle:
00334                             if (i > 0)
00335                             {
00336                                 this.FillText(new Point(metrics.LeftSideBearing, fullMetrics.Bottom +
      fullMetrics.Height / 2), c, font, fillColour, textBaseline:  TextBaselines.Baseline, glyphTag);
00337                             }
00338                             else
00339                             {
00340                                 this.FillText(new Point(0, fullMetrics.Bottom + fullMetrics.Height /
      2), c, font, fillColour, textBaseline:  TextBaselines.Baseline, glyphTag);
00341                             }
00342                             break;
00343                     }
00344 
00345                     this.Restore();
00346 
00347                     if (i > 0)
00348                     {
00349                         currDelta += (metrics.Width + metrics.RightSideBearing +
      metrics.LeftSideBearing + currentGlyphAdvanceDelta.X * font.FontSize / 1000) / pathLength;
00350                     }
00351                     else
00352                     {
00353                         currDelta += (metrics.Width + metrics.RightSideBearing +
      currentGlyphAdvanceDelta.X * font.FontSize / 1000) / pathLength;
00354                     }
00355                 }
00356             }
00357         }
00358 
00359 /// <summary>
00360 /// Stroke a text string along a <see cref="GraphicsPath"/>.
00361 /// </summary>
00362 /// <param name="path">The <see cref="GraphicsPath"/> along which the text will flow.</param>
00363 /// <param name="text">The string to draw.</param>
00364 /// <param name="font">The font with which to draw the text.</param>
00365 /// <param name="strokeColour">The <see cref="Brush"/> with which to stroke the text.</param>
00366 /// <param name="lineWidth">The width of the line with which the text is stroked.</param>
00367 /// <param name="lineCap">The line cap to use to stroke the text.</param>
00368 /// <param name="lineJoin">The line join to use to stroke the text.</param>
00369 /// <param name="lineDash">The line dash to use to stroke the text.</param>
00370 /// <param name="reference">The (relative) starting point on the path starting from which the text
      should be drawn (0 is the start of the path, 1 is the end of the path).</param>
```

```
00371 /// <param name="anchor">The anchor in the text string that will correspond to the point specified by
      the <paramref name="reference"/>.</param>
00372 /// <param name="textBaseline">The text baseline (determines which the position of the text in
      relation to the <paramref name="path"/>.</param>
00373 /// <param name="tag">A tag to identify the stroked text.</param>
00374         public void StrokeTextOnPath(GraphicsPath path, string text, Font font, Brush strokeColour,
      double reference = 0, TextAnchors anchor = TextAnchors.Left, TextBaselines textBaseline =
      TextBaselines.Top, double lineWidth = 1, LineCaps lineCap = LineCaps.Butt, LineJoins lineJoin =
      LineJoins.Miter, LineDash?  lineDash = null, string tag = null)
00375         {
00376             if (!string.IsNullOrEmpty(text))
00377             {
00378                 if (font.Underline != null)
00379                 {
00380                     font = new Font(font.FontFamily, font.FontSize, false);
00381                 }
00382 
00383                 double currDelta = 0;
00384                 double pathLength = path.MeasureLength();
00385 
00386                 Font.DetailedFontMetrics fullMetrics = font.MeasureTextAdvanced(text);
00387 
00388                 switch (anchor)
00389                 {
00390                     case TextAnchors.Left:
00391                         break;
00392                     case TextAnchors.Center:
00393                         currDelta = -fullMetrics.Width * 0.5 / pathLength;
00394                         break;
00395                     case TextAnchors.Right:
00396                         currDelta = -fullMetrics.Width / pathLength;
00397                         break;
00398                 }
00399 
00400                 Point currentGlyphPlacementDelta = new Point();
00401                 Point currentGlyphAdvanceDelta = new Point();
00402                 Point nextGlyphPlacementDelta = new Point();
00403                 Point nextGlyphAdvanceDelta = new Point();
00404 
00405                 for (int i = 0; i < text.Length; i++)
00406                 {
00407                     string c = text.Substring(i, 1);
00408 
00409                     if (Font.EnableKerning && i < text.Length - 1)
00410                     {
00411                         currentGlyphPlacementDelta = nextGlyphPlacementDelta;
00412                         currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
00413                         nextGlyphAdvanceDelta = new Point();
00414                         nextGlyphPlacementDelta = new Point();
00415 
00416                         TrueTypeFile.PairKerning kerning =
      font.FontFamily.TrueTypeFile.Get1000EmKerning(text[i], text[i + 1]);
00417 
00418                         if (kerning != null)
00419                         {
00420                             currentGlyphPlacementDelta = new Point(currentGlyphPlacementDelta.X +
      kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y + kerning.Glyph1Placement.Y);
00421                             currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
      kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
00422 
00423                             nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
      kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
00424                             nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
      kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
00425                         }
00426                     }
00427 
00428                     Font.DetailedFontMetrics metrics = font.MeasureTextAdvanced(c);
00429 
00430                     Point origin = path.GetPointAtRelative(reference + currDelta +
      currentGlyphPlacementDelta.X * font.FontSize / 1000);
00431 
00432                     Point tangent = path.GetTangentAtRelative(reference + currDelta +
      currentGlyphPlacementDelta.X * font.FontSize / 1000 + (metrics.Width + metrics.RightSideBearing +
      metrics.LeftSideBearing) / pathLength * 0.5);
00433 
00434                     origin = new Point(origin.X - tangent.Y * currentGlyphPlacementDelta.Y *
      font.FontSize / 1000, origin.Y + tangent.X * currentGlyphPlacementDelta.Y * font.FontSize / 1000);
00435 
00436                     this.Save();
00437 
00438                     this.Translate(origin);
00439                     this.Rotate(Math.Atan2(tangent.Y, tangent.X));
00440 
00441                     string glyphTag = (!string.IsNullOrEmpty(tag) && UseUniqueTags) ?  (tag + "@" +
      i.ToString()) :  tag;
00442 
```

```
00443                     switch (textBaseline)
00444                     {
00445                         case TextBaselines.Top:
00446                             if (i > 0)
00447                             {
00448                                 this.StrokeText(new Point(metrics.LeftSideBearing, fullMetrics.Top),
      c, font, strokeColour, textBaseline:  TextBaselines.Baseline, lineWidth, lineCap, lineJoin, lineDash,
      glyphTag);
00449                             }
00450                             else
00451                             {
00452                                 this.StrokeText(new Point(0, fullMetrics.Top), c, font, strokeColour,
      textBaseline:  TextBaselines.Baseline, lineWidth, lineCap, lineJoin, lineDash, glyphTag);
00453                             }
00454                             break;
00455                         case TextBaselines.Baseline:
00456                             if (i > 0)
00457                             {
00458                                 this.StrokeText(new Point(metrics.LeftSideBearing, 0), c, font,
      strokeColour, textBaseline:  TextBaselines.Baseline, lineWidth, lineCap, lineJoin, lineDash,
      glyphTag);
00459                             }
00460                             else
00461                             {
00462                                 this.StrokeText(new Point(0, 0), c, font, strokeColour, textBaseline:
      TextBaselines.Baseline, lineWidth, lineCap, lineJoin, lineDash, glyphTag);
00463                             }
00464                             break;
00465                         case TextBaselines.Bottom:
00466                             if (i > 0)
00467                             {
00468                                 this.StrokeText(new Point(metrics.LeftSideBearing,
      fullMetrics.Bottom), c, font, strokeColour, textBaseline:  TextBaselines.Baseline, lineWidth, lineCap,
      lineJoin, lineDash, glyphTag);
00469                             }
00470                             else
00471                             {
00472                                 this.StrokeText(new Point(0, fullMetrics.Bottom), c, font,
      strokeColour, textBaseline:  TextBaselines.Baseline, lineWidth, lineCap, lineJoin, lineDash,
      glyphTag);
00473                             }
00474                             break;
00475                         case TextBaselines.Middle:
00476                             if (i > 0)
00477                             {
00478                                 this.StrokeText(new Point(metrics.LeftSideBearing, fullMetrics.Bottom
      + fullMetrics.Height / 2), c, font, strokeColour, textBaseline:  TextBaselines.Baseline, lineWidth,
      lineCap, lineJoin, lineDash, glyphTag);
00479                             }
00480                             else
00481                             {
00482                                 this.StrokeText(new Point(0, fullMetrics.Bottom + fullMetrics.Height /
      2), c, font, strokeColour, textBaseline:  TextBaselines.Baseline, lineWidth, lineCap, lineJoin,
      lineDash, glyphTag);
00483                             }
00484                             break;
00485                     }
00486 
00487                     this.Restore();
00488 
00489                     if (i > 0)
00490                     {
00491                         currDelta += (metrics.Width + metrics.RightSideBearing +
      metrics.LeftSideBearing + currentGlyphAdvanceDelta.X * font.FontSize / 1000) / pathLength;
00492                     }
00493                     else
00494                     {
00495                         currDelta += (metrics.Width + metrics.RightSideBearing +
      currentGlyphAdvanceDelta.X * font.FontSize / 1000) / pathLength;
00496                     }
00497                 }
00498             }
00499         }
00500 
00501 /// <summary>
00502 /// Fill a formatted text string.
00503 /// </summary>
00504 /// <param name="origin">The text origin.  See <paramref name="textBaseline"/>.</param>
00505 /// <param name="text">The <see cref="FormattedText"/> to draw.</param>
00506 /// <param name="fillColour">The default <see cref="Brush"/> to use to fill the text.  This can be
      overridden by each <paramref name="text"/> element.</param>
00507 /// <param name="textBaseline">The text baseline (determines what the vertical component of <paramref
      name="origin"/> represents).</param>
00508 /// <param name="tag">A tag to identify the filled text.</param>
00509         public void FillText(Point origin, IEnumerable<FormattedText> text, Brush fillColour,
      TextBaselines textBaseline = TextBaselines.Top, string tag = null)
00510         {
```

```
00511             if (text != null)
00512             {
00513                 List<FormattedText> enumeratedText = new List<FormattedText>();
00514                 List<Font.DetailedFontMetrics> allMetrics = new List<Font.DetailedFontMetrics>();
00515 
00516                 Font.DetailedFontMetrics fullMetrics = text.Measure(enumeratedText, allMetrics);
00517 
00518                 if (enumeratedText.Count > 0)
00519                 {
00520                     Point baselineOrigin = origin;
00521 
00522                     switch (textBaseline)
00523                     {
00524                         case TextBaselines.Baseline:
00525                             baselineOrigin = origin;
00526                             break;
00527                         case TextBaselines.Top:
00528                             baselineOrigin = new Point(origin.X, origin.Y + fullMetrics.Top);
00529                             break;
00530                         case TextBaselines.Bottom:
00531                             baselineOrigin = new Point(origin.X, origin.Y + fullMetrics.Bottom);
00532                             break;
00533                         case TextBaselines.Middle:
00534                             baselineOrigin = new Point(origin.X, origin.Y + fullMetrics.Top * 0.5 +
      fullMetrics.Bottom * 0.5);
00535                             break;
00536                     }
00537 
00538                     for (int i = 0; i < enumeratedText.Count; i++)
00539                     {
00540                         FormattedText txt = enumeratedText[i];
00541 
00542                         string spanTag = (!string.IsNullOrEmpty(tag) && UseUniqueTags) ?  (tag + "@" +
      i.ToString()) :  tag;
00543 
00544                         if (txt.Script == Script.Normal)
00545                         {
00546                             Font.DetailedFontMetrics metrics = allMetrics[i];
00547 
00548                             if (i > 0)
00549                             {
00550                                 if (txt.Spacing == TextSpacing.Default)
00551                                 {
00552                                     FillText(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y, txt.Text, txt.Font, txt.Brush ??  fillColour, TextBaselines.Baseline, spanTag);
00553                                 }
00554                                 else
00555                                 {
00556                                     FillTextWithSpacing(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y, txt.Text, txt.Font, txt.Brush ??  fillColour, txt.Spacing, TextBaselines.Baseline,
      spanTag);
00557                                 }
00558                             }
00559                             else
00560                             {
00561                                 if (txt.Spacing == TextSpacing.Default)
00562                                 {
00563                                     FillText(baselineOrigin, txt.Text, txt.Font, txt.Brush ??
      fillColour, TextBaselines.Baseline, spanTag);
00564                                 }
00565                                 else
00566                                 {
00567                                     FillTextWithSpacing(baselineOrigin.X, baselineOrigin.Y, txt.Text,
      txt.Font, txt.Brush ??  fillColour, txt.Spacing, TextBaselines.Baseline, spanTag);
00568                                 }
00569                             }
00570 
00571                             if (i > 0)
00572                             {
00573                                 baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing + metrics.LeftSideBearing, baselineOrigin.Y);
00574                             }
00575                             else
00576                             {
00577                                 baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing, baselineOrigin.Y);
00578                             }
00579                         }
00580                         else
00581                         {
00582                             Font newFont = new Font(txt.Font.FontFamily, txt.Font.FontSize * 0.7,
      txt.Font.Underline);
00583 
00584                             Font.DetailedFontMetrics metrics = allMetrics[i];
00585 
00586                             if (i == 0)
00587                             {
```

```
00588                                   baselineOrigin = new Point(baselineOrigin.X - metrics.LeftSideBearing,
      baselineOrigin.Y);
00589                               }
00590
00591                               if (txt.Script == Script.Subscript)
00592                               {
00593                                   if (txt.Spacing == TextSpacing.Default)
00594                                   {
00595                                       FillText(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y + txt.Font.FontSize * 0.14, txt.Text, newFont, txt.Brush ??  fillColour,
      TextBaselines.Baseline, spanTag);
00596                                   }
00597                                   else
00598                                   {
00599                                       FillTextWithSpacing(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y + txt.Font.FontSize * 0.14, txt.Text, newFont, txt.Brush ??  fillColour, txt.Spacing,
      TextBaselines.Baseline, spanTag);
00600                                   }
00601                               }
00602                               else if (txt.Script == Script.Superscript)
00603                               {
00604                                   if (txt.Spacing == TextSpacing.Default)
00605                                   {
00606                                       FillText(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y - txt.Font.FontSize * 0.33, txt.Text, newFont, txt.Brush ??  fillColour,
      TextBaselines.Baseline, spanTag);
00607                                   }
00608                                   else
00609                                   {
00610                                       FillTextWithSpacing(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y - txt.Font.FontSize * 0.33, txt.Text, newFont, txt.Brush ??  fillColour, txt.Spacing,
      TextBaselines.Baseline, spanTag);
00611                                   }
00612                               }
00613
00614
00615                               if (i > 0)
00616                               {
00617                                   baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing + metrics.LeftSideBearing, baselineOrigin.Y);
00618                               }
00619                               else
00620                               {
00621                                   baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing, baselineOrigin.Y);
00622                               }
00623                           }
00624                       }
00625                   }
00626               }
00627           }
00628
00629 /// <summary>
00630 /// Fill a formatted text string.
00631 /// </summary>
00632 /// <param name="originX">The horizontal coordinate of the text origin.</param>
00633 /// <param name="originY">The vertical coordinate of the text origin.  See <paramref
      name="textBaseline"/>.</param>
00634 /// <param name="text">The <see cref="FormattedText"/> to draw.</param>
00635 /// <param name="fillColour">The default <see cref="Brush"/> to use to fill the text.  This can be
      overridden by each <paramref name="text"/> element.</param>
00636 /// <param name="textBaseline">The text baseline (determines what <paramref name="originY"/>
      represents).</param>
00637 /// <param name="tag">A tag to identify the filled text.</param>
00638         public void FillText(double originX, double originY, IEnumerable<FormattedText> text, Brush
      fillColour, TextBaselines textBaseline = TextBaselines.Top, string tag = null)
00639         {
00640             FillText(new Point(originX, originY), text, fillColour, textBaseline, tag);
00641         }
00642
00643 /// <summary>
00644 /// Stroke a formatted text string.
00645 /// </summary>
00646 /// <param name="origin">The text origin.  See <paramref name="textBaseline"/>.</param>
00647 /// <param name="text">The <see cref="FormattedText"/> to draw.</param>
00648 /// <param name="strokeColour">The default <see cref="Brush"/> with which to stroke the text.</param>
00649 /// <param name="lineWidth">The width of the line with which the text is stroked.</param>
00650 /// <param name="lineCap">The line cap to use to stroke the text.</param>
00651 /// <param name="lineJoin">The line join to use to stroke the text.</param>
00652 /// <param name="lineDash">The line dash to use to stroke the text.</param>
00653 /// <param name="textBaseline">The text baseline (determines what the vertical component of <paramref
      name="origin"/> represents).</param>
00654 /// <param name="tag">A tag to identify the stroked text.</param>
00655         public void StrokeText(Point origin, IEnumerable<FormattedText> text, Brush strokeColour,
      TextBaselines textBaseline = TextBaselines.Top, double lineWidth = 1, LineCaps lineCap =
      LineCaps.Butt, LineJoins lineJoin = LineJoins.Miter, LineDash?  lineDash = null, string tag = null)
00656         {
```

```
00657             if (text != null)
00658             {
00659                 List<FormattedText> enumeratedText = new List<FormattedText>();
00660                 List<Font.DetailedFontMetrics> allMetrics = new List<Font.DetailedFontMetrics>();
00661 
00662                 Font.DetailedFontMetrics fullMetrics = text.Measure(enumeratedText, allMetrics);
00663                 if (enumeratedText.Count > 0)
00664                 {
00665                     Point baselineOrigin = origin;
00666 
00667                     switch (textBaseline)
00668                     {
00669                         case TextBaselines.Baseline:
00670                             baselineOrigin = origin;
00671                             break;
00672                         case TextBaselines.Top:
00673                             baselineOrigin = new Point(origin.X, origin.Y + fullMetrics.Top);
00674                             break;
00675                         case TextBaselines.Bottom:
00676                             baselineOrigin = new Point(origin.X, origin.Y + fullMetrics.Bottom);
00677                             break;
00678                         case TextBaselines.Middle:
00679                             baselineOrigin = new Point(origin.X, origin.Y + fullMetrics.Top * 0.5 +
      fullMetrics.Bottom * 0.5);
00680                             break;
00681                     }
00682 
00683                     for (int i = 0; i < enumeratedText.Count; i++)
00684                     {
00685                         FormattedText txt = enumeratedText[i];
00686 
00687                         string spanTag = (!string.IsNullOrEmpty(tag) && UseUniqueTags) ?  (tag + "@" +
      i.ToString()) :  tag;
00688 
00689                         if (txt.Script == Script.Normal)
00690                         {
00691                             Font.DetailedFontMetrics metrics = allMetrics[i];
00692 
00693                             if (i > 0)
00694                             {
00695                                 if (txt.Spacing == TextSpacing.Default)
00696                                 {
00697                                     StrokeText(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y, txt.Text, txt.Font, txt.Brush ??  strokeColour, TextBaselines.Baseline, lineWidth,
      lineCap, lineJoin, lineDash, spanTag);
00698                                 }
00699                                 else
00700                                 {
00701                                     StrokeTextWithSpacing(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y, txt.Text, txt.Font, txt.Brush ??  strokeColour, txt.Spacing, TextBaselines.Baseline,
      lineWidth, lineCap, lineJoin, lineDash, spanTag);
00702                                 }
00703                             }
00704                             else
00705                             {
00706                                 if (txt.Spacing == TextSpacing.Default)
00707                                 {
00708                                     StrokeText(baselineOrigin, txt.Text, txt.Font, txt.Brush ??
      strokeColour, TextBaselines.Baseline, lineWidth, lineCap, lineJoin, lineDash, spanTag);
00709                                 }
00710                                 else
00711                                 {
00712                                     StrokeTextWithSpacing(baselineOrigin.X, baselineOrigin.Y,
      txt.Text, txt.Font, txt.Brush ??  strokeColour, txt.Spacing, TextBaselines.Baseline, lineWidth,
      lineCap, lineJoin, lineDash, spanTag);
00713                                 }
00714                             }
00715 
00716                             if (i > 0)
00717                             {
00718                                 baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing + metrics.LeftSideBearing, baselineOrigin.Y);
00719                             }
00720                             else
00721                             {
00722                                 baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing, baselineOrigin.Y);
00723                             }
00724                         }
00725                         else
00726                         {
00727                             Font newFont = new Font(txt.Font.FontFamily, txt.Font.FontSize * 0.7,
      txt.Font.Underline);
00728 
00729                             Font.DetailedFontMetrics metrics = allMetrics[i];
00730 
00731                             if (i == 0)
```

```
00732                               {
00733                                   baselineOrigin = new Point(baselineOrigin.X - metrics.LeftSideBearing,
      baselineOrigin.Y);
00734                               }
00735
00736                               if (txt.Script == Script.Subscript)
00737                               {
00738                                   if (txt.Spacing == TextSpacing.Default)
00739                                   {
00740                                       StrokeText(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y + txt.Font.FontSize * 0.14, txt.Text, newFont, txt.Brush ??  strokeColour,
      TextBaselines.Baseline, lineWidth, lineCap, lineJoin, lineDash, spanTag);
00741                                   }
00742                                   else
00743                                   {
00744                                       StrokeTextWithSpacing(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y + txt.Font.FontSize * 0.14, txt.Text, newFont, txt.Brush ??  strokeColour,
      txt.Spacing, TextBaselines.Baseline, lineWidth, lineCap, lineJoin, lineDash, spanTag);
00745                                   }
00746                               }
00747                               else if (txt.Script == Script.Superscript)
00748                               {
00749                                   if (txt.Spacing == TextSpacing.Default)
00750                                   {
00751                                       StrokeText(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y - txt.Font.FontSize * 0.33, txt.Text, newFont, txt.Brush ??  strokeColour,
      TextBaselines.Baseline, lineWidth, lineCap, lineJoin, lineDash, spanTag);
00752                                   }
00753                                   else
00754                                   {
00755                                       StrokeTextWithSpacing(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y - txt.Font.FontSize * 0.33, txt.Text, newFont, txt.Brush ??  strokeColour,
      txt.Spacing, TextBaselines.Baseline, lineWidth, lineCap, lineJoin, lineDash, spanTag);
00756                                   }
00757                               }
00758
00759
00760                               if (i > 0)
00761                               {
00762                                   baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing + metrics.LeftSideBearing, baselineOrigin.Y);
00763                               }
00764                               else
00765                               {
00766                                   baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing, baselineOrigin.Y);
00767                               }
00768                           }
00769                       }
00770                   }
00771               }
00772           }
00773
00774 /// <summary>
00775 /// Stroke a formatted text string.
00776 /// </summary>
00777 /// <param name="originX">The horizontal coordinate of the text origin.</param>
00778 /// <param name="originY">The vertical coordinate of the text origin.  See <paramref
      name="textBaseline"/>.</param>
00779 /// <param name="text">The <see cref="FormattedText"/> to draw.</param>
00780 /// <param name="strokeColour">The default <see cref="Brush"/> with which to stroke the text.</param>
00781 /// <param name="lineWidth">The width of the line with which the text is stroked.</param>
00782 /// <param name="lineCap">The line cap to use to stroke the text.</param>
00783 /// <param name="lineJoin">The line join to use to stroke the text.</param>
00784 /// <param name="lineDash">The line dash to use to stroke the text.</param>
00785 /// <param name="textBaseline">The text baseline (determines what <paramref name="originY"/>
      represents).</param>
00786 /// <param name="tag">A tag to identify the stroked text.</param>
00787         public void StrokeText(double originX, double originY, IEnumerable<FormattedText> text, Brush
      strokeColour, TextBaselines textBaseline = TextBaselines.Top, double lineWidth = 1, LineCaps lineCap =
      LineCaps.Butt, LineJoins lineJoin = LineJoins.Miter, LineDash?  lineDash = null, string tag = null)
00788         {
00789             StrokeText(new Point(originX, originY), text, strokeColour, textBaseline, lineWidth,
      lineCap, lineJoin, lineDash, tag);
00790         }
00791
00792 /// <summary>
00793 /// Measure a text string.
00794 /// See also <seealso cref="Font.MeasureText(string)"/> and <seealso
      cref="Font.MeasureTextAdvanced(string)"/>.
00795 /// </summary>
00796 /// <param name="text">The string to measure.</param>
00797 /// <param name="font">The font to use to measure the string.</param>
00798 /// <returns>The size of the measured <paramref name="text"/>.</returns>
00799         public Size MeasureText(string text, Font font)
00800         {
00801             if (!string.IsNullOrEmpty(text))
```

```
00802             {
00803                 return font.MeasureText(text);
00804             }
00805             else
00806             {
00807                 return new Size(0, 0);
00808             }
00809         }
00810 
00811 /// <summary>
00812 /// Measure a formatted text string.
00813 /// See also <seealso cref="FormattedTextExtensions.Measure(IEnumerable{FormattedText})"/>.
00814 /// </summary>
00815 /// <param name="text">The collection of <see cref="FormattedText"/> objects to measure.</param>
00816 /// <returns>The size of the measured <paramref name="text"/>.</returns>
00817         public Size MeasureText(IEnumerable<FormattedText> text)
00818         {
00819             Font.DetailedFontMetrics metrics = text.Measure();
00820 
00821             return new Size(metrics.Width, metrics.Height);
00822         }
00823 
00824 /// <summary>
00825 /// Fills the underline of the specified text string.
00826 /// </summary>
00827 /// <param name="originX">The horizontal coordinate of the text origin.</param>
00828 /// <param name="originY">The vertical coordinate of the text origin.  See <paramref
      name="textBaseline"/>.</param>
00829 /// <param name="text">The string whose underline will be draw.</param>
00830 /// <param name="font">The font with which to draw the text.</param>
00831 /// <param name="fillColour">The <see cref="Brush"/> to use to fill the underline.</param>
00832 /// <param name="textBaseline">The text baseline (determines what <paramref name="originY"/>
      represents).</param>
00833 /// <param name="tag">A tag to identify the filled underline.</param>
00834         public void FillTextUnderline(double originX, double originY, string text, Font font, Brush
      fillColour, TextBaselines textBaseline = TextBaselines.Top, string tag = null)
00835         {
00836             FillTextUnderline(new Point(originX, originY), text, font, fillColour, textBaseline, tag);
00837         }
00838 
00839 /// <summary>
00840 /// Fills the underline of the specified text string.
00841 /// </summary>
00842 /// <param name="origin">The text origin.  See <paramref name="textBaseline"/>.</param>
00843 /// <param name="text">The string whose underline will be draw.</param>
00844 /// <param name="font">The font with which to draw the text.</param>
00845 /// <param name="fillColour">The <see cref="Brush"/> to use to fill the underline.</param>
00846 /// <param name="textBaseline">The text baseline (determines what the vertical component of <paramref
      name="origin"/> represents).</param>
00847 /// <param name="tag">A tag to identify the filled underline.</param>
00848         public void FillTextUnderline(Point origin, string text, Font font, Brush fillColour,
      TextBaselines textBaseline = TextBaselines.Top, string tag = null)
00849         {
00850             if (!string.IsNullOrEmpty(text))
00851             {
00852                 GraphicsPath underline = new GraphicsPath().AddTextUnderline(origin, text, font,
      textBaseline);
00853                 FillPath(underline, fillColour, tag);
00854             }
00855         }
00856 
00857 /// <summary>
00858 /// Stroke the underline of the specified text string.
00859 /// </summary>
00860 /// <param name="originX">The horizontal coordinate of the text origin.</param>
00861 /// <param name="originY">The vertical coordinate of the text origin.  See <paramref
      name="textBaseline"/>.</param>
00862 /// <param name="text">The string whose underline will be drawn.</param>
00863 /// <param name="font">The font with which to draw the text.</param>
00864 /// <param name="strokeColour">The <see cref="Brush"/> with which to stroke the underline.</param>
00865 /// <param name="lineWidth">The width of the line with which the underline is stroked.</param>
00866 /// <param name="lineCap">The line cap to use to stroke the underline.</param>
00867 /// <param name="lineJoin">The line join to use to stroke the underline.</param>
00868 /// <param name="lineDash">The line dash to use to stroke the underline.</param>
00869 /// <param name="textBaseline">The text baseline (determines what <paramref name="originY"/>
      represents).</param>
00870 /// <param name="tag">A tag to identify the stroked underline.</param>
00871         public void StrokeTextUnderline(double originX, double originY, string text, Font font, Brush
      strokeColour, TextBaselines textBaseline = TextBaselines.Top, double lineWidth = 1, LineCaps lineCap =
      LineCaps.Butt, LineJoins lineJoin = LineJoins.Miter, LineDash?  lineDash = null, string tag = null)
00872         {
00873             StrokeTextUnderline(new Point(originX, originY), text, font, strokeColour, textBaseline,
      lineWidth, lineCap, lineJoin, lineDash, tag);
00874         }
00875 
00876 /// <summary>
00877 /// Stroke the underline of the specified text string.
```

```
00878 /// </summary>
00879 /// <param name="origin">The text origin.  See <paramref name="textBaseline"/>.</param>
00880 /// <param name="text">The string whose underline will be drawn.</param>
00881 /// <param name="font">The font with which to draw the text.</param>
00882 /// <param name="strokeColour">The <see cref="Brush"/> with which to stroke the underline.</param>
00883 /// <param name="lineWidth">The width of the line with which the underline is stroked.</param>
00884 /// <param name="lineCap">The line cap to use to stroke the underline.</param>
00885 /// <param name="lineJoin">The line join to use to stroke the underline.</param>
00886 /// <param name="lineDash">The line dash to use to stroke the underline.</param>
00887 /// <param name="textBaseline">The text baseline (determines what the vertical component of <paramref
      name="origin"/> represents).</param>
00888 /// <param name="tag">A tag to identify the stroked underline.</param>
00889         public void StrokeTextUnderline(Point origin, string text, Font font, Brush strokeColour,
      TextBaselines textBaseline = TextBaselines.Top, double lineWidth = 1, LineCaps lineCap =
      LineCaps.Butt, LineJoins lineJoin = LineJoins.Miter, LineDash?  lineDash = null, string tag = null)
00890         {
00891             if (!string.IsNullOrEmpty(text))
00892             {
00893                 GraphicsPath underline = new GraphicsPath().AddTextUnderline(origin, text, font,
      textBaseline);
00894                 StrokePath(underline, strokeColour, lineWidth, lineCap, lineJoin, lineDash, tag);
00895             }
00896         }
00897
00898
00899 /// <summary>
00900 /// Fill the underline of the specified formatted text string.
00901 /// </summary>
00902 /// <param name="originX">The horizontal coordinate of the text origin.</param>
00903 /// <param name="originY">The vertical coordinate of the text origin.  See <paramref
      name="textBaseline"/>.</param>
00904 /// <param name="text">The <see cref="FormattedText"/> whose underline will be drawn.</param>
00905 /// <param name="fillColour">The default <see cref="Brush"/> to use to fill the underline.  This can
      be overridden by each <paramref name="text"/> element.</param>
00906 /// <param name="textBaseline">The text baseline (determines what <paramref name="originY"/>
      represents).</param>
00907 /// <param name="tag">A tag to identify the filled underlined.</param>
00908         public void FillTextUnderline(double originX, double originY, IEnumerable<FormattedText> text,
      Brush fillColour, TextBaselines textBaseline = TextBaselines.Top, string tag = null)
00909         {
00910             FillTextUnderline(new Point(originX, originY), text, fillColour, textBaseline, tag);
00911         }
00912
00913 /// <summary>
00914 /// Fill the underline of the specified formatted text string.
00915 /// </summary>
00916 /// <param name="origin">The text origin.  See <paramref name="textBaseline"/>.</param>
00917 /// <param name="text">The <see cref="FormattedText"/> whose underline will be drawn.</param>
00918 /// <param name="fillColour">The default <see cref="Brush"/> to use to fill the underline.  This can
      be overridden by each <paramref name="text"/> element.</param>
00919 /// <param name="textBaseline">The text baseline (determines what the vertical component of <paramref
      name="origin"/> represents).</param>
00920 /// <param name="tag">A tag to identify the filled underlined.</param>
00921         public void FillTextUnderline(Point origin, IEnumerable<FormattedText> text, Brush fillColour,
      TextBaselines textBaseline = TextBaselines.Top, string tag = null)
00922         {
00923             if (text != null)
00924             {
00925
00926                 List<FormattedText> enumeratedText = new List<FormattedText>();
00927                 List<Font.DetailedFontMetrics> allMetrics = new List<Font.DetailedFontMetrics>();
00928
00929                 Font.DetailedFontMetrics fullMetrics = text.Measure(enumeratedText, allMetrics);
00930
00931                 if (enumeratedText.Count > 0)
00932                 {
00933
00934                     Point baselineOrigin = origin;
00935
00936                     switch (textBaseline)
00937                     {
00938                         case TextBaselines.Baseline:
00939                             baselineOrigin = origin;
00940                             break;
00941                         case TextBaselines.Top:
00942                             baselineOrigin = new Point(origin.X, origin.Y + fullMetrics.Top);
00943                             break;
00944                         case TextBaselines.Bottom:
00945                             baselineOrigin = new Point(origin.X, origin.Y + fullMetrics.Bottom);
00946                             break;
00947                         case TextBaselines.Middle:
00948                             baselineOrigin = new Point(origin.X, origin.Y + fullMetrics.Top * 0.5 +
      fullMetrics.Bottom * 0.5);
00949                             break;
00950                     }
00951
00952                     for (int i = 0; i < enumeratedText.Count; i++)
```

```
00953                     {
00954                         FormattedText txt = enumeratedText[i];
00955 
00956                         string spanTag = (!string.IsNullOrEmpty(tag) && UseUniqueTags) ?  (tag + "@" +
      i.ToString()) :  tag;
00957 
00958                         if (txt.Script == Script.Normal)
00959                         {
00960                             Font.DetailedFontMetrics metrics = allMetrics[i];
00961 
00962                             if (i > 0)
00963                             {
00964                                 FillTextUnderline(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y, txt.Text, txt.Font, txt.Brush ??  fillColour, TextBaselines.Baseline, spanTag);
00965                             }
00966                             else
00967                             {
00968                                 FillTextUnderline(baselineOrigin, txt.Text, txt.Font, txt.Brush ??
      fillColour, TextBaselines.Baseline, spanTag);
00969                             }
00970 
00971                             if (i > 0)
00972                             {
00973                                 baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing + metrics.LeftSideBearing, baselineOrigin.Y);
00974                             }
00975                             else
00976                             {
00977                                 baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing, baselineOrigin.Y);
00978                             }
00979                         }
00980                         else
00981                         {
00982                             Font newFont = new Font(txt.Font.FontFamily, txt.Font.FontSize * 0.7,
      txt.Font.Underline);
00983 
00984                             Font.DetailedFontMetrics metrics = allMetrics[i];
00985 
00986                             if (i == 0)
00987                             {
00988                                 baselineOrigin = new Point(baselineOrigin.X - metrics.LeftSideBearing,
      baselineOrigin.Y);
00989                             }
00990 
00991                             if (txt.Script == Script.Subscript)
00992                             {
00993                                 FillTextUnderline(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y + txt.Font.FontSize * 0.14, txt.Text, newFont, txt.Brush ??  fillColour,
      TextBaselines.Baseline, spanTag);
00994                             }
00995                             else if (txt.Script == Script.Superscript)
00996                             {
00997                                 FillTextUnderline(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y - txt.Font.FontSize * 0.33, txt.Text, newFont, txt.Brush ??  fillColour,
      TextBaselines.Baseline, spanTag);
00998                             }
00999 
01000 
01001                             if (i > 0)
01002                             {
01003                                 baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing + metrics.LeftSideBearing, baselineOrigin.Y);
01004                             }
01005                             else
01006                             {
01007                                 baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing, baselineOrigin.Y);
01008                             }
01009                         }
01010                     }
01011                 }
01012             }
01013         }
01014 
01015 /// <summary>
01016 /// Stroke the underline of the specified formatted text string.
01017 /// </summary>
01018 /// <param name="originX">The horizontal coordinate of the text origin.</param>
01019 /// <param name="originY">The vertical coordinate of the text origin.  See <paramref
      name="textBaseline"/>.</param>
01020 /// <param name="text">The <see cref="FormattedText"/> to draw.</param>
01021 /// <param name="strokeColour">The default <see cref="Brush"/> with which to stroke the
      underline.</param>
01022 /// <param name="lineWidth">The width of the line with which the underline is stroked.</param>
01023 /// <param name="lineCap">The line cap to use to stroke the underline.</param>
01024 /// <param name="lineJoin">The line join to use to stroke the underline.</param>
```

```
01025 /// <param name="lineDash">The line dash to use to stroke the underline.</param>
01026 /// <param name="textBaseline">The text baseline (determines what <paramref name="originY"/>
      represents).</param>
01027 /// <param name="tag">A tag to identify the stroked underline.</param>
01028         public void StrokeTextUnderline(double originX, double originY, IEnumerable<FormattedText>
      text, Brush strokeColour, TextBaselines textBaseline = TextBaselines.Top, double lineWidth = 1,
      LineCaps lineCap = LineCaps.Butt, LineJoins lineJoin = LineJoins.Miter, LineDash?  lineDash = null,
      string tag = null)
01029         {
01030             StrokeTextUnderline(new Point(originX, originY), text, strokeColour, textBaseline,
      lineWidth, lineCap, lineJoin, lineDash, tag);
01031         }
01032
01033 /// <summary>
01034 /// Stroke the underline of the specified formatted text string.
01035 /// </summary>
01036 /// <param name="origin">The text origin.  See <paramref name="textBaseline"/>.</param>
01037 /// <param name="text">The <see cref="FormattedText"/> to draw.</param>
01038 /// <param name="strokeColour">The default <see cref="Brush"/> with which to stroke the
      underline.</param>
01039 /// <param name="lineWidth">The width of the line with which the underline is stroked.</param>
01040 /// <param name="lineCap">The line cap to use to stroke the underline.</param>
01041 /// <param name="lineJoin">The line join to use to stroke the underline.</param>
01042 /// <param name="lineDash">The line dash to use to stroke the underline.</param>
01043 /// <param name="textBaseline">The text baseline (determines what the vertical component of <paramref
      name="origin"/> represents).</param>
01044 /// <param name="tag">A tag to identify the stroked underline.</param>
01045         public void StrokeTextUnderline(Point origin, IEnumerable<FormattedText> text, Brush
      strokeColour, TextBaselines textBaseline = TextBaselines.Top, double lineWidth = 1, LineCaps lineCap =
      LineCaps.Butt, LineJoins lineJoin = LineJoins.Miter, LineDash?  lineDash = null, string tag = null)
01046         {
01047             if (text != null)
01048             {
01049                 List<FormattedText> enumeratedText = new List<FormattedText>();
01050                 List<Font.DetailedFontMetrics> allMetrics = new List<Font.DetailedFontMetrics>();
01051
01052                 Font.DetailedFontMetrics fullMetrics = text.Measure(enumeratedText, allMetrics);
01053
01054                 if (enumeratedText.Count > 0)
01055                 {
01056
01057                     Point baselineOrigin = origin;
01058
01059                     switch (textBaseline)
01060                     {
01061                         case TextBaselines.Baseline:
01062                             baselineOrigin = origin;
01063                             break;
01064                         case TextBaselines.Top:
01065                             baselineOrigin = new Point(origin.X, origin.Y + fullMetrics.Top);
01066                             break;
01067                         case TextBaselines.Bottom:
01068                             baselineOrigin = new Point(origin.X, origin.Y + fullMetrics.Bottom);
01069                             break;
01070                         case TextBaselines.Middle:
01071                             baselineOrigin = new Point(origin.X, origin.Y + fullMetrics.Top * 0.5 +
      fullMetrics.Bottom * 0.5);
01072                             break;
01073                     }
01074
01075                     for (int i = 0; i < enumeratedText.Count; i++)
01076                     {
01077                         FormattedText txt = enumeratedText[i];
01078
01079                         string spanTag = (!string.IsNullOrEmpty(tag) && UseUniqueTags) ?  (tag + "@" +
      i.ToString()) :  tag;
01080
01081                         if (txt.Script == Script.Normal)
01082                         {
01083                             Font.DetailedFontMetrics metrics = allMetrics[i];
01084
01085                             if (i > 0)
01086                             {
01087                                 StrokeTextUnderline(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y, txt.Text, txt.Font, txt.Brush ??  strokeColour, TextBaselines.Baseline, lineWidth,
      lineCap, lineJoin, lineDash, spanTag);
01088                             }
01089                             else
01090                             {
01091                                 StrokeTextUnderline(baselineOrigin, txt.Text, txt.Font, txt.Brush ??
      strokeColour, TextBaselines.Baseline, lineWidth, lineCap, lineJoin, lineDash, spanTag);
01092                             }
01093
01094                             if (i > 0)
01095                             {
01096                                 baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing + metrics.LeftSideBearing, baselineOrigin.Y);
```

```
01097                               }
01098                               else
01099                               {
01100                                   baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing, baselineOrigin.Y);
01101                               }
01102                           }
01103                           else
01104                           {
01105                               Font newFont = new Font(txt.Font.FontFamily, txt.Font.FontSize * 0.7,
      txt.Font.Underline);
01106
01107                               Font.DetailedFontMetrics metrics = allMetrics[i];
01108
01109                               if (i == 0)
01110                               {
01111                                   baselineOrigin = new Point(baselineOrigin.X - metrics.LeftSideBearing,
      baselineOrigin.Y);
01112                               }
01113
01114                               if (txt.Script == Script.Subscript)
01115                               {
01116                                   StrokeTextUnderline(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y + txt.Font.FontSize * 0.14, txt.Text, newFont, txt.Brush ??  strokeColour,
      TextBaselines.Baseline, lineWidth, lineCap, lineJoin, lineDash, spanTag);
01117                               }
01118                               else if (txt.Script == Script.Superscript)
01119                               {
01120                                   StrokeTextUnderline(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y - txt.Font.FontSize * 0.33, txt.Text, newFont, txt.Brush ??  strokeColour,
      TextBaselines.Baseline, lineWidth, lineCap, lineJoin, lineDash, spanTag);
01121                               }
01122
01123
01124                               if (i > 0)
01125                               {
01126                                   baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing + metrics.LeftSideBearing, baselineOrigin.Y);
01127                               }
01128                               else
01129                               {
01130                                   baselineOrigin = new Point(baselineOrigin.X + metrics.Width +
      metrics.RightSideBearing, baselineOrigin.Y);
01131                               }
01132                           }
01133                       }
01134                   }
01135               }
01136           }
01137
01138
01139           private static Font.DetailedFontMetrics TextWithSpacingBase(double x, double y, string text,
      Font font, TextBaselines textBaseline, TextSpacing spacing, Action<double, double, char, int>
      printAction, bool isLast)
01140           {
01141               if (string.IsNullOrEmpty(text))
01142               {
01143                   return new Font.DetailedFontMetrics(0, 0, 0, 0, 0, 0, 0);
01144               }
01145
01146               Font.DetailedFontMetrics metrics = font.MeasureTextAdvanced(text);
01147
01148               Point baselineOrigin = new Point(x, y);
01149
01150               switch (textBaseline)
01151               {
01152                   case TextBaselines.Baseline:
01153                       baselineOrigin = new Point(x - metrics.LeftSideBearing, y);
01154                       break;
01155                   case TextBaselines.Top:
01156                       baselineOrigin = new Point(x - metrics.LeftSideBearing, y + metrics.Top);
01157                       break;
01158                   case TextBaselines.Bottom:
01159                       baselineOrigin = new Point(x - metrics.LeftSideBearing, y + metrics.Bottom);
01160                       break;
01161                   case TextBaselines.Middle:
01162                       baselineOrigin = new Point(x - metrics.LeftSideBearing, y + (metrics.Top -
      metrics.Bottom) * 0.5 + metrics.Bottom);
01163                       break;
01164               }
01165
01166               Point currentGlyphPlacementDelta = new Point();
01167               Point currentGlyphAdvanceDelta = new Point();
01168               Point nextGlyphPlacementDelta = new Point();
01169               Point nextGlyphAdvanceDelta = new Point();
01170               double scale = 1;
01171               double increment = 0;
```

```
01172             char c = '\0';
01173 
01174             for (int i = 0; i < text.Length; i++)
01175             {
01176                 c = text[i];
01177 
01178                 if (Font.EnableKerning && i < text.Length - 1)
01179                 {
01180                     currentGlyphPlacementDelta = nextGlyphPlacementDelta;
01181                     currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
01182                     nextGlyphAdvanceDelta = new Point();
01183                     nextGlyphPlacementDelta = new Point();
01184 
01185                     TrueTypeFile.PairKerning kerning =
      font.FontFamily.TrueTypeFile.Get1000EmKerning(c, text[i + 1]);
01186 
01187                     if (kerning != null)
01188                     {
01189                         currentGlyphPlacementDelta = new Point(currentGlyphPlacementDelta.X +
      kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y + kerning.Glyph1Placement.Y);
01190                         currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
      kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
01191 
01192                         nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
      kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
01193                         nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
      kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
01194                     }
01195                 }
01196 
01197                 double lsb = font.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(c).LeftSideBearing *
      font.FontSize / 1000;
01198 
01199                 printAction?.Invoke(baselineOrigin.X + currentGlyphPlacementDelta.X + lsb,
      baselineOrigin.Y + currentGlyphPlacementDelta.Y, c, i);
01200 
01201                 if (char.IsWhiteSpace(c))
01202                 {
01203                     scale = spacing.WhitespaceScale;
01204                     increment = spacing.WhitespaceAdd;
01205                 }
01206                 else
01207                 {
01208                     scale = spacing.NonWhitespaceScale;
01209                     increment = spacing.NonWhitespaceAdd;
01210                 }
01211 
01212                 baselineOrigin.X += (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) +
      currentGlyphAdvanceDelta.X) * font.FontSize / 1000 * scale + increment;
01213                 baselineOrigin.Y += currentGlyphAdvanceDelta.Y * font.FontSize / 1000;
01214             }
01215 
01216             double width;
01217 
01218             if (isLast)
01219             {
01220                 width = baselineOrigin.X - x - (currentGlyphAdvanceDelta.X * font.FontSize / 1000 *
      scale + increment) - font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) * font.FontSize / 1000 *
      (scale - 1) - font.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(c).RightSideBearing * font.FontSize
      / 1000;
01221             }
01222             else
01223             {
01224                 width = baselineOrigin.X - x - (currentGlyphAdvanceDelta.X * font.FontSize / 1000 *
      scale + increment) - font.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(c).RightSideBearing *
      font.FontSize / 1000;
01225             }
01226 
01227             return new Font.DetailedFontMetrics(width, metrics.Height, metrics.LeftSideBearing,
      metrics.RightSideBearing, metrics.Top, metrics.Bottom, metrics.AdvanceWidth - metrics.Width + width);
01228         }
01229 
01230 
01231         private void StrokeTextWithSpacing(double x, double y, string text, Font font, Brush brush,
      TextSpacing spacing, TextBaselines textBaseline = TextBaselines.Top, double lineWidth = 1, LineCaps
      lineCap = LineCaps.Butt, LineJoins lineJoin = LineJoins.Miter, LineDash?  lineDash = null, string tag
      = null)
01232         {
01233             TextWithSpacingBase(x, y, text, font, textBaseline, spacing, (cx, cy, c, i) =>
01234             {
01235                 string charTag = tag;
01236 
01237                 if (this.UseUniqueTags && tag != null)
01238                 {
01239                     charTag = tag + "@" + i.ToString();
01240                 }
01241 
```

```
01242                 this.StrokeText(cx, cy, c.ToString(), font, brush, TextBaselines.Baseline, lineWidth,
      lineCap, lineJoin, lineDash, charTag);
01243             }, true);
01244         }
01245 
01246         private void FillTextWithSpacing(double x, double y, string text, Font font, Brush brush,
      TextSpacing spacing, TextBaselines textBaseline = TextBaselines.Top, string tag = null)
01247         {
01248             TextWithSpacingBase(x, y, text, font, textBaseline, spacing, (cx, cy, c, i) =>
01249             {
01250                 string charTag = tag;
01251 
01252                 if (this.UseUniqueTags && tag != null)
01253                 {
01254                     charTag = tag + "@" + i.ToString();
01255                 }
01256 
01257                 this.FillText(cx, cy, c.ToString(), font, brush, TextBaselines.Baseline, charTag);
01258             }, true);
01259         }
01260 
01261         internal static Font.DetailedFontMetrics MeasureTextWithSpacing(double x, double y, string
      text, Font font, bool isLast, TextSpacing spacing, TextBaselines textBaseline = TextBaselines.Top)
01262         {
01263             return TextWithSpacingBase(x, y, text, font, textBaseline, spacing, null, isLast);
01264         }
01265 
01266 
01267     }
01268 }
```

8.85 GraphicsAction.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using VectSharp.Filters;
00019 
00020 namespace VectSharp
00021 {
00022     internal interface IGraphicsAction
00023     {
00024         IGraphicsAction ShallowClone();
00025         string Tag { get; set; }
00026     }
00027 
00028     internal interface IPrintableAction
00029     {
00030         Brush Fill { get; }
00031         Brush Stroke { get; }
00032         double LineWidth { get; }
00033         LineCaps LineCap { get; }
00034         LineJoins LineJoin { get; }
00035         LineDash LineDash { get; }
00036         string Tag { get; set; }
00037         Rectangle GetBounds();
00038     }
00039 
00040     internal class TransformAction :  IGraphicsAction
00041     {
00042         public Point?  Delta { get; } = null;
00043 
00044         public double?  Angle { get; } = null;
00045 
00046         public Size?  Scale { get; } = null;
00047 
00048         public double[,] Matrix { get; } = null;
00049         public string Tag { get; set; }
00050 
```

```
00051         public IGraphicsAction ShallowClone() => (IGraphicsAction)MemberwiseClone();
00052
00053         public TransformAction(Point delta, string tag)
00054         {
00055             this.Delta = delta;
00056             this.Tag = tag;
00057         }
00058
00059         public TransformAction(double angle, string tag)
00060         {
00061             this.Angle = angle;
00062             this.Tag = tag;
00063         }
00064
00065         public TransformAction(Size scale, string tag)
00066         {
00067             this.Scale = scale;
00068             this.Tag = tag;
00069         }
00070
00071         public TransformAction(double[,] matrix, string tag)
00072         {
00073             this.Matrix = matrix;
00074             this.Tag = tag;
00075         }
00076
00077         public double[,] GetMatrix()
00078         {
00079             if (this.Matrix != null)
00080             {
00081                 return this.Matrix;
00082             }
00083             else if (this.Delta != null)
00084             {
00085                 return Graphics.TranslationMatrix(this.Delta.Value.X, this.Delta.Value.Y);
00086             }
00087             else if (this.Angle != null)
00088             {
00089                 return Graphics.RotationMatrix(this.Angle.Value);
00090             }
00091             else if (this.Scale != null)
00092             {
00093                 return Graphics.ScaleMatrix(this.Scale.Value.Width, this.Scale.Value.Height);
00094             }
00095             else
00096             {
00097                 return null;
00098             }
00099         }
00100     }
00101
00102     internal class StateAction :  IGraphicsAction
00103     {
00104         public string Tag { get; set; } = null;
00105
00106         public IGraphicsAction ShallowClone() => (IGraphicsAction)MemberwiseClone();
00107
00108         public enum StateActionTypes
00109         {
00110             Save, Restore
00111         }
00112
00113         public StateActionTypes StateActionType { get; }
00114
00115         public StateAction(StateActionTypes type)
00116         {
00117             this.StateActionType = type;
00118         }
00119     }
00120
00121     internal class TextAction :  IGraphicsAction, IPrintableAction
00122     {
00123         public IGraphicsAction ShallowClone() => (IGraphicsAction)MemberwiseClone();
00124
00125         public Brush Fill { get; }
00126         public Brush Stroke { get; }
00127         public double LineWidth { get; }
00128         public LineCaps LineCap { get; }
00129         public LineJoins LineJoin { get; }
00130         public LineDash LineDash { get; }
00131         public string Tag { get; set; }
00132         public string Text { get; }
00133         public Point Origin { get; }
00134         public TextBaselines TextBaseline { get; }
00135         public Font Font { get; }
00136
00137         public TextAction(Point origin, string text, Font font, TextBaselines textBaseLine, Brush
```

```
      fill, Brush stroke, double lineWidth, LineCaps lineCap, LineJoins lineJoin, LineDash lineDash, string
      tag)
00138         {
00139             this.Origin = origin;
00140             this.Text = text;
00141             this.Font = font;
00142             this.TextBaseline = textBaseLine;
00143             this.Fill = fill;
00144             this.Stroke = stroke;
00145             this.LineCap = lineCap;
00146             this.LineJoin = lineJoin;
00147             this.LineWidth = lineWidth;
00148             this.Tag = tag;
00149             this.LineDash = lineDash;
00150         }
00151 
00152         public Rectangle GetBounds()
00153         {
00154             Font.DetailedFontMetrics metrics = this.Font.MeasureTextAdvanced(this.Text);
00155 
00156             switch (this.TextBaseline)
00157             {
00158                 case TextBaselines.Top:
00159                     return new Rectangle(this.Origin, new Size(metrics.Width, metrics.Height));
00160                 case TextBaselines.Bottom:
00161                     return new Rectangle(this.Origin.X, this.Origin.Y - metrics.Height, metrics.Width,
      metrics.Height);
00162                 case TextBaselines.Middle:
00163                     return new Rectangle(this.Origin.X, this.Origin.Y - metrics.Height * 0.5,
      metrics.Width, metrics.Height);
00164                 case TextBaselines.Baseline:
00165                     return new Rectangle(this.Origin.X, this.Origin.Y - metrics.Top, metrics.Width,
      metrics.Height);
00166                 default:
00167                     throw new System.ArgumentOutOfRangeException(nameof(TextBaseline),
      this.TextBaseline, "Invalid text baseline!");
00168             }
00169         }
00170     }
00171 
00172     internal class RectangleAction :  IGraphicsAction, IPrintableAction
00173     {
00174         public IGraphicsAction ShallowClone() => (IGraphicsAction)MemberwiseClone();
00175         public Brush Fill { get; }
00176         public Brush Stroke { get; }
00177         public double LineWidth { get; }
00178         public LineCaps LineCap { get; }
00179         public LineJoins LineJoin { get; }
00180         public LineDash LineDash { get; }
00181         public string Tag { get; set; }
00182         public Point TopLeft { get; }
00183         public Size Size { get; }
00184 
00185         public RectangleAction(Point topLeft, Size size, Brush fill, Brush stroke, double lineWidth,
      LineCaps lineCap, LineJoins lineJoin, LineDash lineDash, string tag)
00186         {
00187             this.TopLeft = topLeft;
00188             this.Size = size;
00189             this.Fill = fill;
00190             this.Stroke = stroke;
00191             this.LineCap = lineCap;
00192             this.LineJoin = lineJoin;
00193             this.LineWidth = lineWidth;
00194             this.LineDash = lineDash;
00195             this.Tag = tag;
00196         }
00197 
00198         public Rectangle GetBounds()
00199         {
00200             return new Rectangle(this.TopLeft, this.Size);
00201         }
00202     }
00203 
00204     internal class PathAction :  IGraphicsAction, IPrintableAction
00205     {
00206         public IGraphicsAction ShallowClone() => (IGraphicsAction)MemberwiseClone();
00207         public GraphicsPath Path { get; }
00208         public Brush Fill { get; }
00209         public Brush Stroke { get; }
00210         public string Tag { get; set; }
00211         public double LineWidth { get; }
00212         public LineCaps LineCap { get; }
00213         public LineJoins LineJoin { get; }
00214         public LineDash LineDash { get; }
00215         public bool IsClipping { get; }
00216         public FillRule FillRule { get; }
00217         public Rectangle GetBounds()
```

```
00218         {
00219             return this.Path.GetBounds();
00220         }
00221         public PathAction(GraphicsPath path, Brush fill, Brush stroke, double lineWidth, LineCaps
      lineCap, LineJoins lineJoin, LineDash lineDash, string tag, FillRule fillRule, bool isClipping)
00222         {
00223             this.Path = path;
00224             this.Fill = fill;
00225             this.Stroke = stroke;
00226             this.LineCap = lineCap;
00227             this.LineJoin = lineJoin;
00228             this.LineWidth = lineWidth;
00229             this.LineDash = lineDash;
00230             this.Tag = tag;
00231             this.IsClipping = isClipping;
00232             this.FillRule = fillRule;
00233         }
00234     }
00235
00236     internal class RasterImageAction :  IGraphicsAction, IPrintableAction
00237     {
00238         public IGraphicsAction ShallowClone() => (IGraphicsAction)MemberwiseClone();
00239         public Brush Fill { get; }
00240         public Brush Stroke { get; }
00241         public string Tag { get; set; }
00242         public double LineWidth { get; }
00243         public LineCaps LineCap { get; }
00244         public LineJoins LineJoin { get; }
00245         public LineDash LineDash { get; }
00246         public int SourceX { get; }
00247         public int SourceY { get; }
00248         public int SourceWidth { get; }
00249         public int SourceHeight { get; }
00250         public double DestinationX { get; }
00251         public double DestinationY { get; }
00252         public double DestinationWidth { get; }
00253         public double DestinationHeight { get; }
00254         public RasterImage Image { get; }
00255
00256         public RasterImageAction(int sourceX, int sourceY, int sourceWidth, int sourceHeight, double
      destinationX, double destinationY, double destinationWidth, double destinationHeight, RasterImage
      image, string tag)
00257         {
00258             this.SourceX = sourceX;
00259             this.SourceY = sourceY;
00260             this.SourceWidth = sourceWidth;
00261             this.SourceHeight = sourceHeight;
00262
00263             this.DestinationX = destinationX;
00264             this.DestinationY = destinationY;
00265             this.DestinationWidth = destinationWidth;
00266             this.DestinationHeight = destinationHeight;
00267
00268             this.Image = image;
00269             this.Tag = tag;
00270         }
00271
00272         public Rectangle GetBounds()
00273         {
00274             return new Rectangle(this.DestinationX, this.DestinationY, this.DestinationWidth,
      this.DestinationHeight);
00275         }
00276     }
00277
00278     internal class FilteredGraphicsAction :  IGraphicsAction, IPrintableAction
00279     {
00280         public IGraphicsAction ShallowClone() => (IGraphicsAction)MemberwiseClone();
00281         public Brush Fill { get; }
00282         public Brush Stroke { get; }
00283         public string Tag { get; set; }
00284         public double LineWidth { get; }
00285         public LineCaps LineCap { get; }
00286         public LineJoins LineJoin { get; }
00287         public LineDash LineDash { get; }
00288         public int SourceX { get; }
00289         public int SourceY { get; }
00290         public int SourceWidth { get; }
00291         public int SourceHeight { get; }
00292         public double DestinationX { get; }
00293         public double DestinationY { get; }
00294         public double DestinationWidth { get; }
00295         public double DestinationHeight { get; }
00296         public RasterImage Image { get; }
00297         public Graphics Content { get; }
00298         public IFilter Filter { get; }
00299
00300         public FilteredGraphicsAction(Graphics graphics, IFilter filter)
```

```
00301         {
00302             this.Content = graphics;
00303             this.Filter = filter;
00304         }
00305 
00306         public Rectangle GetBounds()
00307         {
00308             Rectangle bounds = this.Content.GetBounds();
00309 
00310             return new Rectangle(bounds.Location.X - this.Filter.TopLeftMargin.X, bounds.Location.Y -
      this.Filter.TopLeftMargin.Y, bounds.Size.Width + this.Filter.TopLeftMargin.X +
      this.Filter.BottomRightMargin.X, bounds.Size.Height + this.Filter.TopLeftMargin.Y +
      this.Filter.BottomRightMargin.Y);
00311         }
00312     }
00313 }
```

8.86 GraphicsPath.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Diagnostics.Contracts;
00021 using System.Linq;
00022 
00023 namespace VectSharp
00024 {
00025 /// <summary>
00026 /// Represents a graphics path that can be filled or stroked.
00027 /// </summary>
00028     public class GraphicsPath
00029     {
00030 /// <summary>
00031 /// The segments that make up the path.
00032 /// </summary>
00033         public List<Segment> Segments { get; set; } = new List<Segment>();
00034 
00035 
00036 /// <summary>
00037 /// Move the current point without tracing a segment from the previous point.
00038 /// </summary>
00039 /// <param name="p">The new point.</param>
00040 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00041         public GraphicsPath MoveTo(Point p)
00042         {
00043             cachedLength = double.NaN;
00044             cachedBounds = Rectangle.NaN;
00045             Segments.Add(new MoveSegment(p));
00046             return this;
00047         }
00048 
00049 /// <summary>
00050 /// Move the current point without tracing a segment from the previous point.
00051 /// </summary>
00052 /// <param name="x">The horizontal coordinate of the new point.</param>
00053 /// <param name="y">The vertical coordinate of the new point.</param>
00054 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00055         public GraphicsPath MoveTo(double x, double y)
00056         {
00057             MoveTo(new Point(x, y));
00058             return this;
00059         }
00060 
00061 /// <summary>
00062 /// Move the current point and trace a segment from the previous point.
00063 /// </summary>
00064 /// <param name="p">The new point.</param>
```

```
00065 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00066         public GraphicsPath LineTo(Point p)
00067         {
00068             cachedLength = double.NaN;
00069             cachedBounds = Rectangle.NaN;
00070 
00071             if (Segments.Count == 0)
00072             {
00073                 Segments.Add(new MoveSegment(p));
00074             }
00075             else
00076             {
00077                 Segments.Add(new LineSegment(p));
00078             }
00079             return this;
00080         }
00081 
00082 /// <summary>
00083 /// Move the current point and trace a segment from the previous point.
00084 /// </summary>
00085 /// <param name="x">The horizontal coordinate of the new point.</param>
00086 /// <param name="y">The vertical coordinate of the new point.</param>
00087 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00088         public GraphicsPath LineTo(double x, double y)
00089         {
00090             LineTo(new Point(x, y));
00091             return this;
00092         }
00093 
00094 /// <summary>
00095 /// Trace an arc segment from a circle with the specified <paramref name="center"/> and <paramref
      name="radius"/>, starting at <paramref name="startAngle"/> and ending at <paramref name="endAngle"/>.
00096 /// The current point is updated to the end point of the arc.
00097 /// </summary>
00098 /// <param name="center">The center of the arc.</param>
00099 /// <param name="radius">The radius of the arc.</param>
00100 /// <param name="startAngle">The start angle (in radians) of the arc.</param>
00101 /// <param name="endAngle">The end angle (in radians) of the arc.</param>
00102 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00103         public GraphicsPath Arc(Point center, double radius, double startAngle, double endAngle)
00104         {
00105             cachedLength = double.NaN;
00106             cachedBounds = Rectangle.NaN;
00107 
00108             if (Segments.Count == 0)
00109             {
00110                 Segments.Add(new MoveSegment(center.X + radius * Math.Cos(startAngle), center.Y +
      radius * Math.Sin(startAngle)));
00111             }
00112             Segments.Add(new ArcSegment(center, radius, startAngle, endAngle));
00113             return this;
00114         }
00115 
00116 /// <summary>
00117 /// Trace an arc segment from a circle with the specified center and <paramref name="radius"/>,
      starting at <paramref name="startAngle"/> and ending at <paramref name="endAngle"/>.
00118 /// The current point is updated to the end point of the arc.
00119 /// </summary>
00120 /// <param name="centerX">The horizontal coordinate of the center of the arc.</param>
00121 /// <param name="centerY">The vertical coordinate of the center of the arc.</param>
00122 /// <param name="radius">The radius of the arc.</param>
00123 /// <param name="startAngle">The start angle (in radians) of the arc.</param>
00124 /// <param name="endAngle">The end angle (in radians) of the arc.</param>
00125 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00126         public GraphicsPath Arc(double centerX, double centerY, double radius, double startAngle,
      double endAngle)
00127         {
00128             Arc(new Point(centerX, centerY), radius, startAngle, endAngle);
00129             return this;
00130         }
00131 
00132 /// <summary>
00133 /// Trace an arc from an ellipse with the specified radii, rotated by <paramref name="axisAngle"/>
      with respect to the x-axis, starting at the current point and ending at the <paramref
      name="endPoint"/>.
00134 /// </summary>
00135 /// <param name="radiusX">The horizontal radius of the ellipse.</param>
00136 /// <param name="radiusY">The vertical radius of the ellipse.</param>
00137 /// <param name="axisAngle">The angle of the horizontal axis of the ellipse with respect to the
      horizontal axis.</param>
00138 /// <param name="largeArc">Determines whether the large or the small arc is drawn.</param>
00139 /// <param name="sweepClockwise">Determines whether the clockwise or anticlockwise arc is
      drawn.</param>
00140 /// <param name="endPoint">The end point of the arc.</param>
00141 /// <returns></returns>
00142         public GraphicsPath EllipticalArc(double radiusX, double radiusY, double axisAngle, bool
      largeArc, bool sweepClockwise, Point endPoint)
```

```
00143         {
00144             if (radiusX == 0 || radiusY == 0)
00145             {
00146                 return this.LineTo(endPoint);
00147             }
00148             else
00149             {
00150                 radiusX = Math.Abs(radiusX);
00151                 radiusY = Math.Abs(radiusY);
00152             }
00153 
00154             double x1 = 0;
00155             double y1 = 0;
00156 
00157             if (this.Segments.Count > 0)
00158             {
00159                 for (int i = this.Segments.Count - 1; i >= 0; i--)
00160                 {
00161                     if (this.Segments[i].Type != SegmentType.Close)
00162                     {
00163                         x1 = this.Segments[i].Point.X;
00164                         y1 = this.Segments[i].Point.Y;
00165                         break;
00166                     }
00167                 }
00168             }
00169 
00170             double x2 = endPoint.X;
00171             double y2 = endPoint.Y;
00172 
00173             double x1P = Math.Cos(axisAngle) * (x1 - x2) * 0.5 + Math.Sin(axisAngle) * (y1 - y2) *
    0.5;
00174 
00175             if (Math.Abs(x1P) < 1e-7)
00176             {
00177                 x1P = 0;
00178             }
00179 
00180             double y1P = -Math.Sin(axisAngle) * (x1 - x2) * 0.5 + Math.Cos(axisAngle) * (y1 - y2) *
    0.5;
00181 
00182             if (Math.Abs(y1P) < 1e-7)
00183             {
00184                 y1P = 0;
00185             }
00186 
00187             double lambda = x1P * x1P / (radiusX * radiusX) + y1P * y1P / (radiusY * radiusY);
00188 
00189             if (lambda > 1)
00190             {
00191                 double sqrtLambda = Math.Sqrt(lambda);
00192                 radiusX *= sqrtLambda;
00193                 radiusY *= sqrtLambda;
00194             }
00195 
00196             double sqrtTerm = (largeArc != sweepClockwise ?  1 :  -1) * Math.Sqrt(Math.Max(0, (radiusX
    * radiusX * radiusY * radiusY - radiusX * radiusX * y1P * y1P - radiusY * radiusY * x1P * x1P) /
    (radiusX * radiusX * y1P * y1P + radiusY * radiusY * x1P * x1P)));
00197 
00198             double cXP = sqrtTerm * radiusX * y1P / radiusY;
00199             double cYP = -sqrtTerm * radiusY * x1P / radiusX;
00200 
00201             double cX = Math.Cos(axisAngle) * cXP - Math.Sin(axisAngle) * cYP + (x1 + x2) * 0.5;
00202             double cY = Math.Sin(axisAngle) * cXP + Math.Cos(axisAngle) * cYP + (y1 + y2) * 0.5;
00203 
00204             double theta1 = AngleVectors(1, 0, (x1P - cXP) / radiusX, (y1P - cYP) / radiusY);
00205             double deltaTheta = AngleVectors((x1P - cXP) / radiusX, (y1P - cYP) / radiusY, (-x1P -
    cXP) / radiusX, (-y1P - cYP) / radiusY) % (2 * Math.PI);
00206 
00207             if (!sweepClockwise && deltaTheta > 0)
00208             {
00209                 deltaTheta -= 2 * Math.PI;
00210             }
00211             else if (sweepClockwise && deltaTheta < 0)
00212             {
00213                 deltaTheta += 2 * Math.PI;
00214             }
00215 
00216             double r = Math.Min(radiusX, radiusY);
00217 
00218             ArcSegment arc = new ArcSegment(0, 0, r, theta1, theta1 + deltaTheta);
00219 
00220             Segment[] segments = arc.ToBezierSegments();
00221 
00222             for (int i = 0; i < segments.Length; i++)
00223             {
00224                 for (int j = 0; j < segments[i].Points.Length; j++)
```

```
00225                 {
00226                     double newX = segments[i].Points[j].X * radiusX / r;
00227                     double newY = segments[i].Points[j].Y * radiusY / r;
00228 
00229                     segments[i].Points[j] = new Point(newX * Math.Cos(axisAngle) - newY *
      Math.Sin(axisAngle) + cX, newX * Math.Sin(axisAngle) + newY * Math.Cos(axisAngle) + cY);
00230                 }
00231             }
00232 
00233             cachedLength = double.NaN;
00234             cachedBounds = Rectangle.NaN;
00235 
00236             this.Segments.AddRange(segments);
00237 
00238             return this;
00239         }
00240 
00241         private static double AngleVectors(double uX, double uY, double vX, double vY)
00242         {
00243             double tbr = Math.Acos((uX * vX + uY * vY) / Math.Sqrt((uX * uX + uY * uY) * (vX * vX + vY
      * vY)));
00244             double sign = Math.Sign(uX * vY - uY * vX);
00245             if (sign != 0)
00246             {
00247                 tbr *= sign;
00248             }
00249             return tbr;
00250         }
00251 
00252 
00253 /// <summary>
00254 /// Trace a cubic Bezier curve from the current point to a destination point, with two control points.
00255 /// The current point is updated to the end point of the Bezier curve.
00256 /// </summary>
00257 /// <param name="control1">The first control point.</param>
00258 /// <param name="control2">The second control point.</param>
00259 /// <param name="endPoint">The destination point.</param>
00260 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00261         public GraphicsPath CubicBezierTo(Point control1, Point control2, Point endPoint)
00262         {
00263             cachedLength = double.NaN;
00264             cachedBounds = Rectangle.NaN;
00265 
00266             if (Segments.Count == 0)
00267             {
00268                 Segments.Add(new MoveSegment(control1));
00269             }
00270             Segments.Add(new CubicBezierSegment(control1, control2, endPoint));
00271             return this;
00272         }
00273 
00274 /// <summary>
00275 /// Trace a cubic Bezier curve from the current point to a destination point, with two control points.
00276 /// The current point is updated to the end point of the Bezier curve.
00277 /// </summary>
00278 /// <param name="control1X">The horizontal coordinate of the first control point.</param>
00279 /// <param name="control1Y">The vertical coordinate of the first control point.</param>
00280 /// <param name="control2X">The horizontal coordinate of the second control point.</param>
00281 /// <param name="control2Y">The vertical coordinate of the second control point.</param>
00282 /// <param name="endPointX">The horizontal coordinate of the destination point.</param>
00283 /// <param name="endPointY">The vertical coordinate of the destination point.</param>
00284 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00285         public GraphicsPath CubicBezierTo(double control1X, double control1Y, double control2X, double
      control2Y, double endPointX, double endPointY)
00286         {
00287             CubicBezierTo(new Point(control1X, control1Y), new Point(control2X, control2Y), new
      Point(endPointX, endPointY));
00288             return this;
00289         }
00290 
00291 /// <summary>
00292 /// Trace a quadratic Bezier curve from the current point to a destination point, with a single
      control point.
00293 /// The current point is updated to the end point of the Bezier curve.
00294 /// </summary>
00295 /// <param name="control">The control point.</param>
00296 /// <param name="endPoint">The destination point.</param>
00297 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00298         public GraphicsPath QuadraticBezierTo(Point control, Point endPoint)
00299         {
00300             Point currentPoint = control;
00301 
00302             if (Segments.Count > 0)
00303             {
00304                 for (int i = Segments.Count - 1; i >= 0; i--)
00305                 {
00306                     if (Segments[i].Type != SegmentType.Close)
```

```
00307                     {
00308                         currentPoint = Segments[i].Point;
00309                         break;
00310                     }
00311                 }
00312             }
00313 
00314             Point control1 = new Point((currentPoint.X + 2 * control.X) / 3, (currentPoint.Y + 2 *
      control.Y) / 3);
00315             Point control2 = new Point((endPoint.X + 2 * control.X) / 3, (endPoint.Y + 2 * control.Y)
      / 3);
00316 
00317             CubicBezierTo(control1, control2, endPoint);
00318 
00319             return this;
00320         }
00321 
00322 /// <summary>
00323 /// Trace a quadratic Bezier curve from the current point to a destination point, with a single
      control point.
00324 /// The current point is updated to the end point of the Bezier curve.
00325 /// </summary>
00326 /// <param name="controlX">The horizontal coordinate of the control point.</param>
00327 /// <param name="controlY">The vertical coordinate of the control point.</param>
00328 /// <param name="endPointX">The horizontal coordinate of the destination point.</param>
00329 /// <param name="endPointY">The vertical coordinate of the destination point.</param>
00330 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00331         public GraphicsPath QuadraticBezierTo(double controlX, double controlY, double endPointX,
      double endPointY)
00332         {
00333             QuadraticBezierTo(new Point(controlX, controlY), new Point(endPointX, endPointY));
00334             return this;
00335         }
00336 
00337 /// <summary>
00338 /// Trace a segment from the current point to the start point of the figure and flag the figure as
      closed.
00339 /// </summary>
00340 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00341         public GraphicsPath Close()
00342         {
00343             cachedLength = double.NaN;
00344             cachedBounds = Rectangle.NaN;
00345             Segments.Add(new CloseSegment());
00346             return this;
00347         }
00348 
00349 /// <summary>
00350 /// Add the contour of a text string to the current path.
00351 /// </summary>
00352 /// <param name="originX">The horizontal coordinate of the text origin.</param>
00353 /// <param name="originY">The vertical coordinate of the text origin.  See <paramref
      name="textBaseline"/>.</param>
00354 /// <param name="text">The string to draw.</param>
00355 /// <param name="font">The font with which to draw the text.</param>
00356 /// <param name="textBaseline">The text baseline (determines what <paramref name="originY"/>
      represents).</param>
00357 /// /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00358         public GraphicsPath AddText(double originX, double originY, string text, Font font,
      TextBaselines textBaseline = TextBaselines.Top)
00359         {
00360             return AddText(new Point(originX, originY), text, font, textBaseline);
00361         }
00362 
00363 /// <summary>
00364 /// Add the contour of a text string to the current path.
00365 /// </summary>
00366 /// <param name="origin">The text origin.  See <paramref name="textBaseline"/>.</param>
00367 /// <param name="text">The string to draw.</param>
00368 /// <param name="font">The font with which to draw the text.</param>
00369 /// <param name="textBaseline">The text baseline (determines what the vertical component of <paramref
      name="origin"/> represents).</param>
00370 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00371         public GraphicsPath AddText(Point origin, string text, Font font, TextBaselines textBaseline =
      TextBaselines.Top)
00372         {
00373             if (!string.IsNullOrEmpty(text))
00374             {
00375                 Font.DetailedFontMetrics metrics = font.MeasureTextAdvanced(text);
00376 
00377                 Point baselineOrigin = origin;
00378 
00379                 switch (textBaseline)
00380                 {
00381                     case TextBaselines.Baseline:
00382                         baselineOrigin = new Point(origin.X - metrics.LeftSideBearing, origin.Y);
00383                         break;
```

```
00384                     case TextBaselines.Top:
00385                         baselineOrigin = new Point(origin.X - metrics.LeftSideBearing, origin.Y +
      metrics.Top);
00386                         break;
00387                     case TextBaselines.Bottom:
00388                         baselineOrigin = new Point(origin.X - metrics.LeftSideBearing, origin.Y +
      metrics.Bottom);
00389                         break;
00390                     case TextBaselines.Middle:
00391                         baselineOrigin = new Point(origin.X - metrics.LeftSideBearing, origin.Y +
      (metrics.Top - metrics.Bottom) * 0.5 + metrics.Bottom);
00392                         break;
00393                 }
00394
00395                 Point currentGlyphPlacementDelta = new Point();
00396                 Point currentGlyphAdvanceDelta = new Point();
00397                 Point nextGlyphPlacementDelta = new Point();
00398                 Point nextGlyphAdvanceDelta = new Point();
00399
00400                 for (int i = 0; i < text.Length; i++)
00401                 {
00402                     char c = text[i];
00403
00404                     if (Font.EnableKerning && i < text.Length - 1)
00405                     {
00406                         currentGlyphPlacementDelta = nextGlyphPlacementDelta;
00407                         currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
00408                         nextGlyphAdvanceDelta = new Point();
00409                         nextGlyphPlacementDelta = new Point();
00410
00411                         TrueTypeFile.PairKerning kerning =
      font.FontFamily.TrueTypeFile.Get1000EmKerning(c, text[i + 1]);
00412
00413                         if (kerning != null)
00414                         {
00415                             currentGlyphPlacementDelta = new Point(currentGlyphPlacementDelta.X +
      kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y + kerning.Glyph1Placement.Y);
00416                             currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
      kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
00417
00418                             nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
      kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
00419                             nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
      kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
00420                         }
00421                     }
00422
00423                     TrueTypeFile.TrueTypePoint[][] glyphPaths =
      font.FontFamily.TrueTypeFile.GetGlyphPath(c, font.FontSize);
00424
00425                     for (int j = 0; j < glyphPaths.Length; j++)
00426                     {
00427                         for (int k = 0; k < glyphPaths[j].Length; k++)
00428                         {
00429                             if (k == 0)
00430                             {
00431                                 this.MoveTo(glyphPaths[j][k].X + baselineOrigin.X +
      currentGlyphPlacementDelta.X, -glyphPaths[j][k].Y + baselineOrigin.Y + currentGlyphPlacementDelta.Y);
00432                             }
00433                             else
00434                             {
00435                                 if (glyphPaths[j][k].IsOnCurve)
00436                                 {
00437                                     this.LineTo(glyphPaths[j][k].X + baselineOrigin.X +
      currentGlyphPlacementDelta.X, -glyphPaths[j][k].Y + baselineOrigin.Y + currentGlyphPlacementDelta.Y);
00438                                 }
00439                                 else
00440                                 {
00441                                     Point startPoint = this.Segments.Last().Point;
00442                                     Point quadCtrl = new Point(glyphPaths[j][k].X + baselineOrigin.X +
      currentGlyphPlacementDelta.X, -glyphPaths[j][k].Y + baselineOrigin.Y + currentGlyphPlacementDelta.Y);
00443                                     Point endPoint = new Point(glyphPaths[j][k + 1].X +
      baselineOrigin.X + currentGlyphPlacementDelta.X, -glyphPaths[j][k + 1].Y + baselineOrigin.Y +
      currentGlyphPlacementDelta.Y);
00444
00445
00446                                     Point ctrl1 = new Point(startPoint.X / 3 + 2 * quadCtrl.X / 3,
      startPoint.Y / 3 + 2 * quadCtrl.Y / 3);
00447                                     Point ctrl2 = new Point(endPoint.X / 3 + 2 * quadCtrl.X / 3,
      endPoint.Y / 3 + 2 * quadCtrl.Y / 3);
00448
00449                                     this.CubicBezierTo(ctrl1, ctrl2, endPoint);
00450
00451                                     k++;
00452                                 }
00453                             }
00454                         }
```

```
00455
00456                       this.Close();
00457                   }
00458
00459                   baselineOrigin.X += (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) +
      currentGlyphAdvanceDelta.X) * font.FontSize / 1000;
00460                   baselineOrigin.Y += (currentGlyphAdvanceDelta.Y) * font.FontSize / 1000;
00461               }
00462           }
00463           return this;
00464       }
00465
00466 /// <summary>
00467 /// Add the contour of a text string flowing along a <see cref="GraphicsPath"/> to the current path.
00468 /// </summary>
00469 /// <param name="path">The <see cref="GraphicsPath"/> along which the text will flow.</param>
00470 /// <param name="text">The string to draw.</param>
00471 /// <param name="font">The font with which to draw the text.</param>
00472 /// <param name="reference">The (relative) starting point on the path starting from which the text
      should be drawn (0 is the start of the path, 1 is the end of the path).</param>
00473 /// <param name="anchor">The anchor in the text string that will correspond to the point specified by
      the <paramref name="reference"/>.</param>
00474 /// <param name="textBaseline">The text baseline (determines which the position of the text in
      relation to the <paramref name="path"/>.</param>
00475 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00476       public GraphicsPath AddTextOnPath(GraphicsPath path, string text, Font font, double reference
      = 0, TextAnchors anchor = TextAnchors.Left, TextBaselines textBaseline = TextBaselines.Top)
00477       {
00478           if (!string.IsNullOrEmpty(text))
00479           {
00480               cachedLength = double.NaN;
00481               cachedBounds = Rectangle.NaN;
00482
00483               double currDelta = 0;
00484               double pathLength = path.MeasureLength();
00485
00486               Font.DetailedFontMetrics fullMetrics = font.MeasureTextAdvanced(text);
00487
00488               switch (anchor)
00489               {
00490                   case TextAnchors.Left:
00491                       break;
00492                   case TextAnchors.Center:
00493                       currDelta = -fullMetrics.Width * 0.5 / pathLength;
00494                       break;
00495                   case TextAnchors.Right:
00496                       currDelta = -fullMetrics.Width / pathLength;
00497                       break;
00498               }
00499
00500               Point currentGlyphPlacementDelta = new Point();
00501               Point currentGlyphAdvanceDelta = new Point();
00502               Point nextGlyphPlacementDelta = new Point();
00503               Point nextGlyphAdvanceDelta = new Point();
00504
00505               for (int i = 0; i < text.Length; i++)
00506               {
00507                   string c = text.Substring(i, 1);
00508
00509                   if (Font.EnableKerning && i < text.Length - 1)
00510                   {
00511                       currentGlyphPlacementDelta = nextGlyphPlacementDelta;
00512                       currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
00513                       nextGlyphAdvanceDelta = new Point();
00514                       nextGlyphPlacementDelta = new Point();
00515
00516                       TrueTypeFile.PairKerning kerning =
      font.FontFamily.TrueTypeFile.Get1000EmKerning(text[i], text[i + 1]);
00517
00518                       if (kerning != null)
00519                       {
00520                           currentGlyphPlacementDelta = new Point(currentGlyphPlacementDelta.X +
      kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y + kerning.Glyph1Placement.Y);
00521                           currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
      kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
00522
00523                           nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
      kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
00524                           nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
      kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
00525                       }
00526                   }
00527
00528                   Font.DetailedFontMetrics metrics = font.MeasureTextAdvanced(c);
00529
00530                   Point origin = path.GetPointAtRelative(reference + currDelta +
      currentGlyphPlacementDelta.X * font.FontSize / 1000);
```

```
00531
00532                     Point tangent = path.GetTangentAtRelative(reference + currDelta +
      currentGlyphPlacementDelta.X * font.FontSize / 1000 + (metrics.Width + metrics.RightSideBearing +
      metrics.LeftSideBearing) / pathLength * 0.5);
00533
00534                     origin = new Point(origin.X - tangent.Y * currentGlyphPlacementDelta.Y *
      font.FontSize / 1000, origin.Y + tangent.X * currentGlyphPlacementDelta.Y * font.FontSize / 1000);
00535
00536                     GraphicsPath glyphPath = new GraphicsPath();
00537
00538                     switch (textBaseline)
00539                     {
00540                         case TextBaselines.Top:
00541                             if (i > 0)
00542                             {
00543                                 glyphPath.AddText(new Point(metrics.LeftSideBearing, fullMetrics.Top),
      c, font, textBaseline:  TextBaselines.Baseline);
00544                             }
00545                             else
00546                             {
00547                                 glyphPath.AddText(new Point(0, fullMetrics.Top), c, font,
      textBaseline:  TextBaselines.Baseline);
00548                             }
00549                             break;
00550                         case TextBaselines.Baseline:
00551                             if (i > 0)
00552                             {
00553                                 glyphPath.AddText(new Point(metrics.LeftSideBearing, 0), c, font,
      textBaseline:  TextBaselines.Baseline);
00554                             }
00555                             else
00556                             {
00557                                 glyphPath.AddText(new Point(0, 0), c, font, textBaseline:
      TextBaselines.Baseline);
00558                             }
00559                             break;
00560                         case TextBaselines.Bottom:
00561                             if (i > 0)
00562                             {
00563                                 glyphPath.AddText(new Point(metrics.LeftSideBearing,
      fullMetrics.Bottom), c, font, textBaseline:  TextBaselines.Baseline);
00564                             }
00565                             else
00566                             {
00567                                 glyphPath.AddText(new Point(0, fullMetrics.Bottom), c, font,
      textBaseline:  TextBaselines.Baseline);
00568                             }
00569                             break;
00570                         case TextBaselines.Middle:
00571                             if (i > 0)
00572                             {
00573                                 glyphPath.AddText(new Point(metrics.LeftSideBearing,
      fullMetrics.Bottom + fullMetrics.Height / 2), c, font, textBaseline:  TextBaselines.Baseline);
00574                             }
00575                             else
00576                             {
00577                                 glyphPath.AddText(new Point(0, fullMetrics.Bottom + fullMetrics.Height
      / 2), c, font, textBaseline:  TextBaselines.Baseline);
00578                             }
00579                             break;
00580                     }
00581
00582                     double angle = Math.Atan2(tangent.Y, tangent.X);
00583
00584                     for (int j = 0; j < glyphPath.Segments.Count; j++)
00585                     {
00586                         if (glyphPath.Segments[j].Points != null)
00587                         {
00588                             for (int k = 0; k < glyphPath.Segments[j].Points.Length; k++)
00589                             {
00590                                 double newX = glyphPath.Segments[j].Points[k].X * Math.Cos(angle) -
      glyphPath.Segments[j].Points[k].Y * Math.Sin(angle) + origin.X;
00591                                 double newY = glyphPath.Segments[j].Points[k].X * Math.Sin(angle) +
      glyphPath.Segments[j].Points[k].Y * Math.Cos(angle) + origin.Y;
00592
00593                                 glyphPath.Segments[j].Points[k] = new Point(newX, newY);
00594                             }
00595                         }
00596
00597                         this.Segments.Add(glyphPath.Segments[j]);
00598                     }
00599
00600                     if (i > 0)
00601                     {
00602                         currDelta += (metrics.Width + metrics.RightSideBearing +
      metrics.LeftSideBearing + currentGlyphAdvanceDelta.X * font.FontSize / 1000) / pathLength;
00603                     }
```

```
00604                     else
00605                     {
00606                         currDelta += (metrics.Width + metrics.RightSideBearing +
      currentGlyphAdvanceDelta.X * font.FontSize / 1000) / pathLength;
00607                     }
00608                 }
00609             }
00610 
00611             return this;
00612         }
00613 
00614 
00615 
00616 /// <summary>
00617 /// Add the contour of the underline of the specified text string to the current path.
00618 /// </summary>
00619 /// <param name="origin">The text origin.  See <paramref name="textBaseline"/>.</param>
00620 /// <param name="text">The string whose underline will be drawn.</param>
00621 /// <param name="font">The font with which to draw the text.</param>
00622 /// <param name="textBaseline">The text baseline (determines what the vertical component of <paramref
      name="origin"/> represents).</param>
00623 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
00624         public GraphicsPath AddTextUnderline(Point origin, string text, Font font, TextBaselines
      textBaseline = TextBaselines.Top)
00625         {
00626             if (!string.IsNullOrEmpty(text))
00627             {
00628                 if (font.Underline == null)
00629                 {
00630                     return this;
00631                 }
00632 
00633                 Font.DetailedFontMetrics metrics = font.MeasureTextAdvanced(text);
00634 
00635                 double italicAngle = font.FontFamily.TrueTypeFile?.GetItalicAngle() ??  0;
00636 
00637                 if (double.IsNaN(italicAngle))
00638                 {
00639                     italicAngle = 0;
00640                 }
00641 
00642                 Point baselineOrigin = origin;
00643 
00644                 switch (textBaseline)
00645                 {
00646                     case TextBaselines.Baseline:
00647                         baselineOrigin = new Point(origin.X - metrics.LeftSideBearing, origin.Y);
00648                         break;
00649                     case TextBaselines.Top:
00650                         baselineOrigin = new Point(origin.X - metrics.LeftSideBearing, origin.Y +
      metrics.Top);
00651                         break;
00652                     case TextBaselines.Bottom:
00653                         baselineOrigin = new Point(origin.X - metrics.LeftSideBearing, origin.Y +
      metrics.Bottom);
00654                         break;
00655                     case TextBaselines.Middle:
00656                         baselineOrigin = new Point(origin.X - metrics.LeftSideBearing, origin.Y +
      (metrics.Top - metrics.Bottom) * 0.5 + metrics.Bottom);
00657                         break;
00658                 }
00659 
00660                 if (!font.Underline.SkipDescenders)
00661                 {
00662                     double italicShift;
00663 
00664                     if (!font.Underline.FollowItalicAngle || italicAngle == 0)
00665                     {
00666                         italicShift = 0;
00667                     }
00668                     else
00669                     {
00670                         italicShift = font.Underline.Thickness * font.FontSize * Math.Tan(italicAngle
      / 180.0 * Math.PI);
00671                     }
00672 
00673                     if (font.Underline.LineCap == LineCaps.Butt)
00674                     {
00675                         if (!font.Underline.FollowItalicAngle || italicAngle == 0)
00676                         {
00677                             this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing, baselineOrigin.Y +
      font.Underline.Position * font.FontSize);
00678                             this.LineTo(baselineOrigin.X + metrics.LeftSideBearing + metrics.Width,
      baselineOrigin.Y + font.Underline.Position * font.FontSize);
00679                             this.LineTo(baselineOrigin.X + metrics.LeftSideBearing + metrics.Width,
      baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness *
      font.FontSize);
```

```
00680                           this.LineTo(baselineOrigin.X + metrics.LeftSideBearing, baselineOrigin.Y +
    font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize);
00681                           this.Close();
00682                       }
00683                       else
00684                       {
00685                           this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing, baselineOrigin.Y +
    font.Underline.Position * font.FontSize);
00686                           this.LineTo(baselineOrigin.X + metrics.LeftSideBearing + metrics.Width,
    baselineOrigin.Y + font.Underline.Position * font.FontSize);
00687                           this.LineTo(baselineOrigin.X + metrics.LeftSideBearing + metrics.Width +
    italicShift, baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness *
    font.FontSize);
00688                           this.LineTo(baselineOrigin.X + metrics.LeftSideBearing + italicShift,
    baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness *
    font.FontSize);
00689                           this.Close();
00690                       }
00691                   }
00692                   else if (font.Underline.LineCap == LineCaps.Square)
00693                   {
00694                       if (!font.Underline.FollowItalicAngle || italicAngle == 0)
00695                       {
00696                           this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing -
    font.Underline.Thickness * font.FontSize * 0.5, baselineOrigin.Y + font.Underline.Position *
    font.FontSize);
00697                           this.LineTo(baselineOrigin.X + metrics.LeftSideBearing + metrics.Width +
    font.Underline.Thickness * font.FontSize * 0.5, baselineOrigin.Y + font.Underline.Position *
    font.FontSize);
00698                           this.LineTo(baselineOrigin.X + metrics.LeftSideBearing + metrics.Width +
    font.Underline.Thickness * font.FontSize * 0.5, baselineOrigin.Y + font.Underline.Position *
    font.FontSize + font.Underline.Thickness * font.FontSize);
00699                           this.LineTo(baselineOrigin.X + metrics.LeftSideBearing -
    font.Underline.Thickness * font.FontSize * 0.5, baselineOrigin.Y + font.Underline.Position *
    font.FontSize + font.Underline.Thickness * font.FontSize);
00700                           this.Close();
00701                       }
00702                       else
00703                       {
00704                           this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing -
    font.Underline.Thickness * font.FontSize * 0.5, baselineOrigin.Y + font.Underline.Position *
    font.FontSize);
00705                           this.LineTo(baselineOrigin.X + metrics.LeftSideBearing + metrics.Width +
    font.Underline.Thickness * font.FontSize * 0.5, baselineOrigin.Y + font.Underline.Position *
    font.FontSize);
00706                           this.LineTo(baselineOrigin.X + metrics.LeftSideBearing + metrics.Width +
    font.Underline.Thickness * font.FontSize * 0.5 + italicShift, baselineOrigin.Y +
    font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize);
00707                           this.LineTo(baselineOrigin.X + metrics.LeftSideBearing -
    font.Underline.Thickness * font.FontSize * 0.5 + italicShift, baselineOrigin.Y +
    font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize);
00708                           this.Close();
00709                       }
00710                   }
00711                   else if (font.Underline.LineCap == LineCaps.Round)
00712                   {
00713                       if (!font.Underline.FollowItalicAngle || italicAngle == 0)
00714                       {
00715                           this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing, baselineOrigin.Y +
    font.Underline.Position * font.FontSize);
00716                           this.LineTo(baselineOrigin.X + metrics.LeftSideBearing + metrics.Width,
    baselineOrigin.Y + font.Underline.Position * font.FontSize);
00717                           this.Arc(baselineOrigin.X + metrics.LeftSideBearing + metrics.Width,
    baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize
    * 0.5, font.Underline.Thickness * font.FontSize * 0.5, -Math.PI / 2, Math.PI / 2);
00718                           this.LineTo(baselineOrigin.X + metrics.LeftSideBearing, baselineOrigin.Y +
    font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize);
00719                           this.Arc(baselineOrigin.X + metrics.LeftSideBearing, baselineOrigin.Y +
    font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize * 0.5,
    font.Underline.Thickness * font.FontSize * 0.5, Math.PI / 2, 3 * Math.PI / 2);
00720                           this.Close();
00721                       }
00722                       else
00723                       {
00724                           this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing - italicShift,
    baselineOrigin.Y + font.Underline.Position * font.FontSize);
00725                           this.LineTo(baselineOrigin.X + metrics.LeftSideBearing + metrics.Width,
    baselineOrigin.Y + font.Underline.Position * font.FontSize);
00726                           this.CubicBezierTo(baselineOrigin.X + metrics.LeftSideBearing +
    metrics.Width + font.Underline.Thickness * font.FontSize, baselineOrigin.Y + font.Underline.Position *
    font.FontSize,
00727                               baselineOrigin.X + metrics.LeftSideBearing + metrics.Width +
    italicShift + font.Underline.Thickness * font.FontSize, baselineOrigin.Y + font.Underline.Position *
    font.FontSize + font.Underline.Thickness * font.FontSize,
00728                               baselineOrigin.X + metrics.LeftSideBearing + metrics.Width +
    italicShift, baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness *
    font.FontSize);
```

```
00729
00730                           this.LineTo(baselineOrigin.X + metrics.LeftSideBearing, baselineOrigin.Y +
     font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize);
00731                           this.CubicBezierTo(baselineOrigin.X + metrics.LeftSideBearing -
     font.Underline.Thickness * font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize +
     font.Underline.Thickness * font.FontSize,
00732                                   baselineOrigin.X + metrics.LeftSideBearing - italicShift -
     font.Underline.Thickness * font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize,
00733                                   baselineOrigin.X + metrics.LeftSideBearing - italicShift,
     baselineOrigin.Y + font.Underline.Position * font.FontSize);
00734
00735                           this.Close();
00736                       }
00737                   }
00738
00739                   return this;
00740               }
00741               else
00742               {
00743                   if (font.Underline.LineCap == LineCaps.Butt)
00744                   {
00745                       double italicShift;
00746
00747                       if (!font.Underline.FollowItalicAngle || italicAngle == 0)
00748                       {
00749                           italicShift = 0;
00750                       }
00751                       else
00752                       {
00753                           italicShift = font.Underline.Thickness * font.FontSize *
     Math.Tan(italicAngle / 180.0 * Math.PI);
00754                       }
00755
00756                       bool started = false;
00757
00758                       double currX = baselineOrigin.X;
00759                       double underlineStartX = baselineOrigin.X + metrics.LeftSideBearing;
00760                       double currUnderlineX = underlineStartX - metrics.LeftSideBearing;
00761
00762                       Point currentGlyphPlacementDelta = new Point();
00763                       Point currentGlyphAdvanceDelta = new Point();
00764                       Point nextGlyphPlacementDelta = new Point();
00765                       Point nextGlyphAdvanceDelta = new Point();
00766
00767                       for (int i = 0; i < text.Length; i++)
00768                       {
00769                           char c = text[i];
00770
00771                           if (Font.EnableKerning && i < text.Length - 1)
00772                           {
00773                               currentGlyphPlacementDelta = nextGlyphPlacementDelta;
00774                               currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
00775                               nextGlyphAdvanceDelta = new Point();
00776                               nextGlyphPlacementDelta = new Point();
00777
00778                               TrueTypeFile.PairKerning kerning =
     font.FontFamily.TrueTypeFile.Get1000EmKerning(c, text[i + 1]);
00779
00780                               if (kerning != null)
00781                               {
00782                                   currentGlyphPlacementDelta = new
     Point(currentGlyphPlacementDelta.X + kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y +
     kerning.Glyph1Placement.Y);
00783                                   currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
     kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
00784
00785                                   nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
     kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
00786                                   nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
     kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
00787                               }
00788                           }
00789
00790                           double[] intersections =
     font.FontFamily.TrueTypeFile.Get1000EmUnderlineIntersections(c, font.Underline.Position * 1000,
     font.Underline.Thickness * 1000);
00791
00792                           if (intersections != null)
00793                           {
00794                               intersections[0] = intersections[0] * font.FontSize / 1000;
00795                               intersections[1] = intersections[1] * font.FontSize / 1000;
00796
00797                               if (currX + intersections[0] - font.Underline.Thickness *
     font.FontSize >= underlineStartX)
00798                               {
00799                                   if (!started)
00800                                   {
```

```
00801                                       started = true;
00802                                       this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing +
      italicShift, baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness *
      font.FontSize).LineTo(baselineOrigin.X + metrics.LeftSideBearing, baselineOrigin.Y +
      font.Underline.Position * font.FontSize);
00803                                   }
00804
00805                                   this.LineTo(currX + intersections[0] - font.Underline.Thickness *
      font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize);
00806                                   this.LineTo(currX + intersections[0] - font.Underline.Thickness *
      font.FontSize + italicShift, baselineOrigin.Y + font.Underline.Position * font.FontSize +
      font.Underline.Thickness * font.FontSize);
00807                                   this.Close();
00808                               }
00809
00810                               started = true;
00811
00812                               this.MoveTo(currX + intersections[1] + font.Underline.Thickness *
      font.FontSize + italicShift, baselineOrigin.Y + font.Underline.Position * font.FontSize +
      font.Underline.Thickness * font.FontSize);
00813                               this.LineTo(currX + intersections[1] + font.Underline.Thickness *
      font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize);
00814
00815                               underlineStartX = currX + intersections[1] + font.Underline.Thickness
      * font.FontSize;
00816                               currUnderlineX = Math.Max(currX + intersections[1] +
      font.Underline.Thickness * font.FontSize, currX + (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c)
      + currentGlyphAdvanceDelta.X) * font.FontSize / 1000 -
      font.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(c).RightSideBearing * font.FontSize / 1000);
00817                           }
00818                           else if (i == text.Length - 1)
00819                           {
00820                               if (c != ' ')
00821                               {
00822                                   currUnderlineX +=
      (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) + currentGlyphAdvanceDelta.X) * font.FontSize /
      1000 - font.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(c).RightSideBearing * font.FontSize / 1000;
00823                               }
00824                               else
00825                               {
00826                                   currUnderlineX +=
      (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) + currentGlyphAdvanceDelta.X) * font.FontSize /
      1000;
00827                               }
00828                           }
00829                           else
00830                           {
00831                               currUnderlineX += (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c)
      + currentGlyphAdvanceDelta.X) * font.FontSize / 1000;
00832                           }
00833
00834                           currX += (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) +
      currentGlyphAdvanceDelta.X) * font.FontSize / 1000;
00835                       }
00836
00837                       if (!started)
00838                       {
00839                           started = true;
00840                           this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing + italicShift,
      baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness *
      font.FontSize).LineTo(baselineOrigin.X + metrics.LeftSideBearing, baselineOrigin.Y +
      font.Underline.Position * font.FontSize);
00841                       }
00842
00843                       this.LineTo(currUnderlineX, baselineOrigin.Y + font.Underline.Position *
      font.FontSize);
00844                       this.LineTo(currUnderlineX + italicShift, baselineOrigin.Y +
      font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize);
00845                       this.LineTo(underlineStartX + italicShift, baselineOrigin.Y +
      font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize);
00846                       this.Close();
00847                   }
00848                   else if (font.Underline.LineCap == LineCaps.Square)
00849                   {
00850                       double italicShift;
00851
00852                       if (!font.Underline.FollowItalicAngle || italicAngle == 0)
00853                       {
00854                           italicShift = 0;
00855                       }
00856                       else
00857                       {
00858                           italicShift = font.Underline.Thickness * font.FontSize *
      Math.Tan(italicAngle / 180.0 * Math.PI);
00859                       }
00860
00861                       bool started = false;
```

```
00862
00863                           double currX = baselineOrigin.X;
00864                           double underlineStartX = baselineOrigin.X + metrics.LeftSideBearing;
00865                           double currUnderlineX = underlineStartX - metrics.LeftSideBearing;
00866
00867                           Point currentGlyphPlacementDelta = new Point();
00868                           Point currentGlyphAdvanceDelta = new Point();
00869                           Point nextGlyphPlacementDelta = new Point();
00870                           Point nextGlyphAdvanceDelta = new Point();
00871
00872                           for (int i = 0; i < text.Length; i++)
00873                           {
00874                               char c = text[i];
00875
00876                               if (Font.EnableKerning && i < text.Length - 1)
00877                               {
00878                                   currentGlyphPlacementDelta = nextGlyphPlacementDelta;
00879                                   currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
00880                                   nextGlyphAdvanceDelta = new Point();
00881                                   nextGlyphPlacementDelta = new Point();
00882
00883                                   TrueTypeFile.PairKerning kerning =
      font.FontFamily.TrueTypeFile.Get1000EmKerning(c, text[i + 1]);
00884
00885                                   if (kerning != null)
00886                                   {
00887                                       currentGlyphPlacementDelta = new
      Point(currentGlyphPlacementDelta.X + kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y +
      kerning.Glyph1Placement.Y);
00888                                       currentGlyphAdvanceDelta = new Point(currentGlyphAdvanceDelta.X +
      kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y + kerning.Glyph1Advance.Y);
00889
00890                                       nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X +
      kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
00891                                       nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
      kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
00892                                   }
00893                               }
00894
00895                               double[] intersections =
      font.FontFamily.TrueTypeFile.Get1000EmUnderlineIntersections(c, font.Underline.Position * 1000,
      font.Underline.Thickness * 1000);
00896
00897                               if (intersections != null)
00898                               {
00899                                   intersections[0] = intersections[0] * font.FontSize / 1000;
00900                                   intersections[1] = intersections[1] * font.FontSize / 1000;
00901
00902                                   if (currX + intersections[0] - font.Underline.Thickness *
      font.FontSize >= underlineStartX)
00903                                   {
00904                                       if (!started)
00905                                       {
00906                                           started = true;
00907                                           this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing +
      italicShift - font.Underline.Thickness * font.FontSize * 0.5, baselineOrigin.Y +
      font.Underline.Position * font.FontSize + font.Underline.Thickness *
      font.FontSize).LineTo(baselineOrigin.X + metrics.LeftSideBearing - font.Underline.Thickness *
      font.FontSize * 0.5, baselineOrigin.Y + font.Underline.Position * font.FontSize);
00908                                       }
00909                                       this.LineTo(currX + intersections[0] - font.Underline.Thickness *
      font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize);
00910                                       this.LineTo(currX + intersections[0] - font.Underline.Thickness *
      font.FontSize + italicShift, baselineOrigin.Y + font.Underline.Position * font.FontSize +
      font.Underline.Thickness * font.FontSize);
00911
00912                                       this.Close();
00913                                   }
00914
00915                                   started = true;
00916
00917                                   this.MoveTo(currX + intersections[1] + font.Underline.Thickness *
      font.FontSize + italicShift, baselineOrigin.Y + font.Underline.Position * font.FontSize +
      font.Underline.Thickness * font.FontSize);
00918                                   this.LineTo(currX + intersections[1] + font.Underline.Thickness *
      font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize);
00919
00920                                   underlineStartX = currX + intersections[1] + font.Underline.Thickness
      * font.FontSize;
00921                                   currUnderlineX = Math.Max(currX + intersections[1] +
      font.Underline.Thickness * font.FontSize, currX + (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c)
      + currentGlyphAdvanceDelta.X) * font.FontSize / 1000 -
      font.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(c).RightSideBearing * font.FontSize / 1000);
00922                               }
00923                               else if (i == text.Length - 1)
00924                               {
00925                                   if (c != ' ')
```

```
00926                                 {
00927                                     currUnderlineX +=
      (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) + currentGlyphAdvanceDelta.X) * font.FontSize /
      1000 - font.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(c).RightSideBearing * font.FontSize / 1000;
00928                                 }
00929                                 else
00930                                 {
00931                                     currUnderlineX +=
      (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) + currentGlyphAdvanceDelta.X) * font.FontSize /
      1000;
00932                                 }
00933                             }
00934                             else
00935                             {
00936                                 currUnderlineX += (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c)
      + currentGlyphAdvanceDelta.X) * font.FontSize / 1000;
00937                             }
00938 
00939                             currX += (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) +
      currentGlyphAdvanceDelta.X) * font.FontSize / 1000;
00940                         }
00941 
00942                         if (!started)
00943                         {
00944                             started = true;
00945                             this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing + italicShift -
      font.Underline.Thickness * font.FontSize * 0.5, baselineOrigin.Y + font.Underline.Position *
      font.FontSize + font.Underline.Thickness * font.FontSize).LineTo(baselineOrigin.X +
      metrics.LeftSideBearing - font.Underline.Thickness * font.FontSize * 0.5, baselineOrigin.Y +
      font.Underline.Position * font.FontSize);
00946                         }
00947 
00948                         this.LineTo(currUnderlineX + font.Underline.Thickness * font.FontSize * 0.5,
      baselineOrigin.Y + font.Underline.Position * font.FontSize);
00949                         this.LineTo(currUnderlineX + italicShift + font.Underline.Thickness *
      font.FontSize * 0.5, baselineOrigin.Y + font.Underline.Position * font.FontSize +
      font.Underline.Thickness * font.FontSize);
00950                         this.LineTo(underlineStartX + italicShift, baselineOrigin.Y +
      font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize);
00951                         this.Close();
00952                     }
00953                     else if (font.Underline.LineCap == LineCaps.Round)
00954                     {
00955                         if (!font.Underline.FollowItalicAngle || italicAngle == 0)
00956                         {
00957                             bool started = false;
00958 
00959                             double currX = baselineOrigin.X;
00960                             double underlineStartX = baselineOrigin.X + metrics.LeftSideBearing;
00961                             double currUnderlineX = underlineStartX - metrics.LeftSideBearing;
00962 
00963                             Point currentGlyphPlacementDelta = new Point();
00964                             Point currentGlyphAdvanceDelta = new Point();
00965                             Point nextGlyphPlacementDelta = new Point();
00966                             Point nextGlyphAdvanceDelta = new Point();
00967 
00968                             for (int i = 0; i < text.Length; i++)
00969                             {
00970                                 char c = text[i];
00971 
00972                                 if (Font.EnableKerning && i < text.Length - 1)
00973                                 {
00974                                     currentGlyphPlacementDelta = nextGlyphPlacementDelta;
00975                                     currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
00976                                     nextGlyphAdvanceDelta = new Point();
00977                                     nextGlyphPlacementDelta = new Point();
00978 
00979                                     TrueTypeFile.PairKerning kerning =
      font.FontFamily.TrueTypeFile.Get1000EmKerning(c, text[i + 1]);
00980 
00981                                     if (kerning != null)
00982                                     {
00983                                         currentGlyphPlacementDelta = new
      Point(currentGlyphPlacementDelta.X + kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y +
      kerning.Glyph1Placement.Y);
00984                                         currentGlyphAdvanceDelta = new
      Point(currentGlyphAdvanceDelta.X + kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y +
      kerning.Glyph1Advance.Y);
00985 
00986                                         nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X
      + kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
00987                                         nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
      kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
00988                                     }
00989                                 }
00990 
00991                                 double[] intersections =
```

```
      font.FontFamily.TrueTypeFile.Get1000EmUnderlineIntersections(c, font.Underline.Position * 1000,
      font.Underline.Thickness * 1000);
00992
00993                                 if (intersections != null)
00994                                 {
00995                                     intersections[0] = intersections[0] * font.FontSize / 1000;
00996                                     intersections[1] = intersections[1] * font.FontSize / 1000;
00997
00998                                     if (currX + intersections[0] - font.Underline.Thickness *
      font.FontSize >= underlineStartX)
00999                                     {
01000                                         if (!started)
01001                                         {
01002                                             started = true;
01003                                             this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness *
      font.FontSize);
01004                                             this.Arc(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize
      * 0.5, font.Underline.Thickness * font.FontSize * 0.5, Math.PI / 2, 3 * Math.PI / 2);
01005                                         }
01006
01007                                         this.LineTo(currX + intersections[0] -
      font.Underline.Thickness * font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize);
01008                                         this.LineTo(currX + intersections[0] -
      font.Underline.Thickness * font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize +
      font.Underline.Thickness * font.FontSize);
01009
01010                                         this.Close();
01011                                     }
01012
01013                                     started = true;
01014
01015                                     this.MoveTo(currX + intersections[1] + font.Underline.Thickness *
      font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness *
      font.FontSize);
01016                                     this.LineTo(currX + intersections[1] + font.Underline.Thickness *
      font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize);
01017
01018                                     underlineStartX = currX + intersections[1] +
      font.Underline.Thickness * font.FontSize;
01019                                     currUnderlineX = Math.Max(currX + intersections[1] +
      font.Underline.Thickness * font.FontSize, currX + (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c)
      + currentGlyphAdvanceDelta.X) * font.FontSize / 1000 -
      font.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(c).RightSideBearing * font.FontSize / 1000);
01020                                 }
01021                                 else if (i == text.Length - 1)
01022                                 {
01023                                     if (c != ' ')
01024                                     {
01025                                         currUnderlineX +=
      (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) + currentGlyphAdvanceDelta.X) * font.FontSize /
      1000 - font.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(c).RightSideBearing * font.FontSize / 1000;
01026                                     }
01027                                     else
01028                                     {
01029                                         currUnderlineX +=
      (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) + currentGlyphAdvanceDelta.X) * font.FontSize /
      1000;
01030                                     }
01031                                 }
01032                                 else
01033                                 {
01034                                     currUnderlineX +=
      (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) + currentGlyphAdvanceDelta.X) * font.FontSize /
      1000;
01035                                 }
01036
01037                                 currX += (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) +
      currentGlyphAdvanceDelta.X) * font.FontSize / 1000;
01038                             }
01039
01040                             if (!started)
01041                             {
01042                                 started = true;
01043                                 this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness *
      font.FontSize);
01044                                 this.Arc(baselineOrigin.X + metrics.LeftSideBearing, baselineOrigin.Y
      + font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize * 0.5,
      font.Underline.Thickness * font.FontSize * 0.5, Math.PI / 2, 3 * Math.PI / 2);
01045                             }
01046
01047                             this.LineTo(currUnderlineX, baselineOrigin.Y + font.Underline.Position *
      font.FontSize);
01048
01049                             this.Arc(currUnderlineX, baselineOrigin.Y + font.Underline.Position *
```

```
    font.FontSize + font.Underline.Thickness * font.FontSize * 0.5, font.Underline.Thickness *
    font.FontSize * 0.5, -Math.PI / 2, Math.PI / 2);
01050
01051                           this.LineTo(underlineStartX, baselineOrigin.Y + font.Underline.Position *
    font.FontSize + font.Underline.Thickness * font.FontSize);
01052                           this.Close();
01053                       }
01054                       else
01055                       {
01056                           double italicShift = font.Underline.Thickness * font.FontSize *
    Math.Tan(italicAngle / 180.0 * Math.PI);
01057
01058                           bool started = false;
01059
01060                           double currX = baselineOrigin.X;
01061                           double underlineStartX = baselineOrigin.X + metrics.LeftSideBearing;
01062                           double currUnderlineX = underlineStartX - metrics.LeftSideBearing;
01063
01064                           Point currentGlyphPlacementDelta = new Point();
01065                           Point currentGlyphAdvanceDelta = new Point();
01066                           Point nextGlyphPlacementDelta = new Point();
01067                           Point nextGlyphAdvanceDelta = new Point();
01068
01069                           for (int i = 0; i < text.Length; i++)
01070                           {
01071                               char c = text[i];
01072
01073                               if (Font.EnableKerning && i < text.Length - 1)
01074                               {
01075                                   currentGlyphPlacementDelta = nextGlyphPlacementDelta;
01076                                   currentGlyphAdvanceDelta = nextGlyphAdvanceDelta;
01077                                   nextGlyphAdvanceDelta = new Point();
01078                                   nextGlyphPlacementDelta = new Point();
01079
01080                                   TrueTypeFile.PairKerning kerning =
    font.FontFamily.TrueTypeFile.Get1000EmKerning(c, text[i + 1]);
01081
01082                                   if (kerning != null)
01083                                   {
01084                                       currentGlyphPlacementDelta = new
    Point(currentGlyphPlacementDelta.X + kerning.Glyph1Placement.X, currentGlyphPlacementDelta.Y +
    kerning.Glyph1Placement.Y);
01085                                       currentGlyphAdvanceDelta = new
    Point(currentGlyphAdvanceDelta.X + kerning.Glyph1Advance.X, currentGlyphAdvanceDelta.Y +
    kerning.Glyph1Advance.Y);
01086
01087                                       nextGlyphPlacementDelta = new Point(nextGlyphPlacementDelta.X
    + kerning.Glyph2Placement.X, nextGlyphPlacementDelta.Y + kerning.Glyph2Placement.Y);
01088                                       nextGlyphAdvanceDelta = new Point(nextGlyphAdvanceDelta.X +
    kerning.Glyph2Advance.X, nextGlyphAdvanceDelta.Y + kerning.Glyph2Advance.Y);
01089                                   }
01090                               }
01091
01092                               double[] intersections =
    font.FontFamily.TrueTypeFile.Get1000EmUnderlineIntersections(c, font.Underline.Position * 1000,
    font.Underline.Thickness * 1000);
01093
01094                               if (intersections != null)
01095                               {
01096                                   intersections[0] = intersections[0] * font.FontSize / 1000;
01097                                   intersections[1] = intersections[1] * font.FontSize / 1000;
01098
01099                                   if (currX + intersections[0] - font.Underline.Thickness *
    font.FontSize >= underlineStartX)
01100                                   {
01101                                       if (!started)
01102                                       {
01103                                           started = true;
01104
01105                                           this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing,
    baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness *
    font.FontSize);
01106
01107                                           this.CubicBezierTo(baselineOrigin.X +
    metrics.LeftSideBearing - font.Underline.Thickness * font.FontSize, baselineOrigin.Y +
    font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize,
01108                                                              baselineOrigin.X + metrics.LeftSideBearing -
    italicShift - font.Underline.Thickness * font.FontSize, baselineOrigin.Y + font.Underline.Position *
    font.FontSize,
01109                                                              baselineOrigin.X + metrics.LeftSideBearing -
    italicShift, baselineOrigin.Y + font.Underline.Position * font.FontSize);
01110                                       }
01111
01112                                       this.LineTo(currX + intersections[0] -
    font.Underline.Thickness * font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize);
01113                                       this.LineTo(currX + intersections[0] -
    font.Underline.Thickness * font.FontSize + italicShift, baselineOrigin.Y + font.Underline.Position *
```

```
      font.FontSize + font.Underline.Thickness * font.FontSize);
01114
01115                                     this.Close();
01116                                 }
01117
01118                                 started = true;
01119
01120                                 this.MoveTo(currX + intersections[1] + font.Underline.Thickness *
      font.FontSize + italicShift, baselineOrigin.Y + font.Underline.Position * font.FontSize +
      font.Underline.Thickness * font.FontSize);
01121                                 this.LineTo(currX + intersections[1] + font.Underline.Thickness *
      font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize);
01122
01123                                 underlineStartX = currX + intersections[1] +
      font.Underline.Thickness * font.FontSize;
01124                                 currUnderlineX = Math.Max(currX + intersections[1] +
      font.Underline.Thickness * font.FontSize, currX + (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c)
      + currentGlyphAdvanceDelta.X) * font.FontSize / 1000 -
      font.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(c).RightSideBearing * font.FontSize / 1000);
01125                             }
01126                             else if (i == text.Length - 1)
01127                             {
01128                                 if (c != ' ')
01129                                 {
01130                                     currUnderlineX +=
      (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) + currentGlyphAdvanceDelta.X) * font.FontSize /
      1000 - font.FontFamily.TrueTypeFile.Get1000EmGlyphBearings(c).RightSideBearing * font.FontSize / 1000;
01131                                 }
01132                                 else
01133                                 {
01134                                     currUnderlineX +=
      (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) + currentGlyphAdvanceDelta.X) * font.FontSize /
      1000;
01135                                 }
01136                             }
01137                             else
01138                             {
01139                                 currUnderlineX +=
      (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) + currentGlyphAdvanceDelta.X) * font.FontSize /
      1000;
01140                             }
01141
01142                             currX += (font.FontFamily.TrueTypeFile.Get1000EmGlyphWidth(c) +
      currentGlyphAdvanceDelta.X) * font.FontSize / 1000;
01143                         }
01144
01145                         if (!started)
01146                         {
01147                             started = true;
01148
01149                             this.MoveTo(baselineOrigin.X + metrics.LeftSideBearing,
      baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness *
      font.FontSize);
01150
01151                             this.CubicBezierTo(baselineOrigin.X + metrics.LeftSideBearing -
      font.Underline.Thickness * font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize +
      font.Underline.Thickness * font.FontSize,
01152                                 baselineOrigin.X + metrics.LeftSideBearing - italicShift -
      font.Underline.Thickness * font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize,
01153                                 baselineOrigin.X + metrics.LeftSideBearing - italicShift,
      baselineOrigin.Y + font.Underline.Position * font.FontSize);
01154                         }
01155
01156                         this.LineTo(currUnderlineX, baselineOrigin.Y + font.Underline.Position *
      font.FontSize);
01157
01158                         //this.LineTo(currUnderlineX + italicShift + font.Underline.Thickness *
      font.FontSize * 0.5, baselineOrigin.Y + font.Underline.Position * font.FontSize +
      font.Underline.Thickness * font.FontSize);
01159                         this.CubicBezierTo(currUnderlineX + font.Underline.Thickness *
      font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize,
01160                             currUnderlineX + italicShift + font.Underline.Thickness *
      font.FontSize, baselineOrigin.Y + font.Underline.Position * font.FontSize + font.Underline.Thickness *
      font.FontSize,
01161                             currUnderlineX + italicShift, baselineOrigin.Y +
      font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize);
01162
01163                         this.LineTo(underlineStartX + italicShift, baselineOrigin.Y +
      font.Underline.Position * font.FontSize + font.Underline.Thickness * font.FontSize);
01164                         this.Close();
01165                     }
01166                 }
01167             }
01168         }
01169         return this;
01170     }
01171
```

```
01172
01173 /// <summary>
01174 /// Adds a smooth spline composed of cubic bezier segments that pass through the specified points.
01175 /// </summary>
01176 /// <param name="points">The points through which the spline should pass.</param>
01177 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
01178         public GraphicsPath AddSmoothSpline(params Point[] points)
01179         {
01180             if (points.Length == 0)
01181             {
01182                 return this;
01183             }
01184             else if (points.Length == 1)
01185             {
01186                 return this.LineTo(points[0]);
01187             }
01188             else if (points.Length == 2)
01189             {
01190                 return this.LineTo(points[0]).LineTo(points[1]);
01191             }
01192
01193             Point[] smoothedSpline = SmoothSpline.SmoothSplines(points);
01194
01195             this.LineTo(smoothedSpline[0]);
01196
01197             for (int i = 1; i < smoothedSpline.Length; i += 3)
01198             {
01199                 this.CubicBezierTo(smoothedSpline[i], smoothedSpline[i + 1], smoothedSpline[i + 2]);
01200             }
01201
01202             return this;
01203         }
01204
01205 /// <summary>
01206 /// Adds another <see cref="GraphicsPath"/> to the current <see cref="GraphicsPath"/>.
01207 /// </summary>
01208 /// <param name="path">The existing <see cref="GraphicsPath"/> that should be added to the current
      <see cref="GraphicsPath"/>.</param>
01209 /// <returns>The <see cref="GraphicsPath"/>, to allow for chained calls.</returns>
01210         public GraphicsPath AddPath(GraphicsPath path)
01211         {
01212             this.Segments.AddRange(path.Segments);
01213             return this;
01214         }
01215
01216         private double cachedLength = double.NaN;
01217
01218 /// <summary>
01219 /// Measures the length of the <see cref="GraphicsPath"/>.
01220 /// </summary>
01221 /// <returns>The length of the <see cref="GraphicsPath"/></returns>
01222         public double MeasureLength()
01223         {
01224             if (double.IsNaN(cachedLength))
01225             {
01226                 cachedLength = 0;
01227                 Point currPoint = new Point();
01228                 Point figureStartPoint = new Point();
01229
01230                 for (int i = 0; i < this.Segments.Count; i++)
01231                 {
01232                     switch (this.Segments[i].Type)
01233                     {
01234                         case SegmentType.Move:
01235                             currPoint = this.Segments[i].Point;
01236                             figureStartPoint = this.Segments[i].Point;
01237                             break;
01238                         case SegmentType.Line:
01239                             if (i > 0)
01240                             {
01241                                 cachedLength += this.Segments[i].Measure(currPoint);
01242                                 currPoint = this.Segments[i].Point;
01243                             }
01244                             else
01245                             {
01246                                 currPoint = this.Segments[i].Point;
01247                                 figureStartPoint = this.Segments[i].Point;
01248                             }
01249                             break;
01250                         case SegmentType.Arc:
01251                             if (i > 0)
01252                             {
01253                                 cachedLength += this.Segments[i].Measure(currPoint);
01254                                 currPoint = this.Segments[i].Point;
01255                             }
01256                             else
01257                             {
```

```
01258                                   ArcSegment seg = (ArcSegment)this.Segments[i];
01259                                   figureStartPoint = new Point(seg.Points[0].X +
    Math.Cos(seg.StartAngle) * seg.Radius, seg.Points[0].Y + Math.Sin(seg.StartAngle) * seg.Radius);
01260                                   cachedLength += this.Segments[i].Measure(figureStartPoint);
01261                                   currPoint = this.Segments[i].Point;
01262                               }
01263                               break;
01264                           case SegmentType.Close:
01265                               cachedLength += Math.Sqrt((currPoint.X - figureStartPoint.X) *
    (currPoint.X - figureStartPoint.X) + (currPoint.Y - figureStartPoint.Y) * (currPoint.Y -
    figureStartPoint.Y));
01266                               currPoint = figureStartPoint;
01267                               break;
01268                           case SegmentType.CubicBezier:
01269                               if (i > 0)
01270                               {
01271                                   cachedLength += this.Segments[i].Measure(currPoint);
01272                                   currPoint = this.Segments[i].Point;
01273                               }
01274                               else
01275                               {
01276                                   currPoint = this.Segments[i].Points[0];
01277                                   figureStartPoint = this.Segments[i].Points[0];
01278                                   cachedLength += this.Segments[i].Measure(currPoint);
01279                                   currPoint = this.Segments[i].Point;
01280                               }
01281                               break;
01282                       }
01283                   }
01284               }
01285
01286               return cachedLength;
01287           }
01288
01289 /// <summary>
01290 /// Gets the point at the relative position specified on the <see cref="GraphicsPath"/>.
01291 /// </summary>
01292 /// <param name="position">The position on the <see cref="GraphicsPath"/> (0 is the start of the path,
    1 is the end of the path).</param>
01293 /// <returns>The point at the specified position.</returns>
01294         public Point GetPointAtRelative(double position)
01295         {
01296             return GetPointAtAbsolute(position * this.MeasureLength());
01297         }
01298
01299 /// <summary>
01300 /// Gets the point at the absolute position specified on the <see cref="GraphicsPath"/>.
01301 /// </summary>
01302 /// <param name="length">The distance to the point from the start of the <see
    cref="GraphicsPath"/>.</param>
01303 /// <returns>The point at the specified position.</returns>
01304         public Point GetPointAtAbsolute(double length)
01305         {
01306             double pathLength = this.MeasureLength();
01307
01308             if (length >= 0 && length <= pathLength)
01309             {
01310                 double currLen = 0;
01311
01312                 Point currPoint = new Point();
01313                 Point figureStartPoint = new Point();
01314
01315                 for (int i = 0; i < this.Segments.Count; i++)
01316                 {
01317                     switch (this.Segments[i].Type)
01318                     {
01319                         case SegmentType.Move:
01320                             currPoint = this.Segments[i].Point;
01321                             figureStartPoint = this.Segments[i].Point;
01322                             break;
01323                         case SegmentType.Line:
01324                             if (i > 0)
01325                             {
01326                                 double segLength = this.Segments[i].Measure(currPoint);
01327
01328                                 if (currLen + segLength < length)
01329                                 {
01330                                     currLen += segLength;
01331                                     currPoint = this.Segments[i].Point;
01332                                 }
01333                                 else if (segLength > 0)
01334                                 {
01335                                     double pos = (length - currLen) / segLength;
01336                                     return this.Segments[i].GetPointAt(currPoint, pos);
01337                                 }
01338                             }
01339                             else
```

```
01340                               {
01341                                   currPoint = this.Segments[i].Point;
01342                                   figureStartPoint = this.Segments[i].Point;
01343                               }
01344                               break;
01345                           case SegmentType.Arc:
01346                               if (i > 0)
01347                               {
01348                                   double segLength = this.Segments[i].Measure(currPoint);
01349 
01350                                   if (currLen + segLength < length)
01351                                   {
01352                                       currLen += segLength;
01353                                       currPoint = this.Segments[i].Point;
01354                                   }
01355                                   else if (segLength > 0)
01356                                   {
01357                                       double pos = (length - currLen) / segLength;
01358                                       return this.Segments[i].GetPointAt(currPoint, pos);
01359                                   }
01360                               }
01361                               else
01362                               {
01363                                   ArcSegment seg = (ArcSegment)this.Segments[i];
01364                                   figureStartPoint = new Point(seg.Points[0].X +
      Math.Cos(seg.StartAngle) * seg.Radius, seg.Points[0].Y + Math.Sin(seg.StartAngle) * seg.Radius);
01365                                   currPoint = figureStartPoint;
01366 
01367                                   double segLength = this.Segments[i].Measure(currPoint);
01368 
01369                                   if (currLen + segLength < length)
01370                                   {
01371                                       currLen += segLength;
01372                                       currPoint = this.Segments[i].Point;
01373                                   }
01374                                   else if (segLength > 0)
01375                                   {
01376                                       double pos = (length - currLen) / segLength;
01377                                       return this.Segments[i].GetPointAt(currPoint, pos);
01378                                   }
01379                               }
01380                               break;
01381                           case SegmentType.Close:
01382                               {
01383                                   double segLength = Math.Sqrt((currPoint.X - figureStartPoint.X) *
      (currPoint.X - figureStartPoint.X) + (currPoint.Y - figureStartPoint.Y) * (currPoint.Y -
      figureStartPoint.Y));
01384 
01385                                   if (currLen + segLength < length)
01386                                   {
01387                                       currLen += segLength;
01388                                       currPoint = figureStartPoint;
01389                                   }
01390                                   else
01391                                   {
01392                                       double pos = (length - currLen) / segLength;
01393                                       return new Point(currPoint.X * (1 - pos) + figureStartPoint.X *
      pos, currPoint.Y * (1 - pos) + figureStartPoint.Y * pos);
01394                                   }
01395                               }
01396                               break;
01397                           case SegmentType.CubicBezier:
01398                               if (i > 0)
01399                               {
01400                                   double segLength = this.Segments[i].Measure(currPoint);
01401 
01402                                   if (currLen + segLength < length)
01403                                   {
01404                                       currLen += segLength;
01405                                       currPoint = this.Segments[i].Point;
01406                                   }
01407                                   else if (segLength > 0)
01408                                   {
01409                                       double pos = (length - currLen) / segLength;
01410                                       return this.Segments[i].GetPointAt(currPoint, pos);
01411                                   }
01412                               }
01413                               else
01414                               {
01415                                   currPoint = this.Segments[i].Points[0];
01416                                   figureStartPoint = this.Segments[i].Points[0];
01417                                   double segLength = this.Segments[i].Measure(currPoint);
01418 
01419                                   if (currLen + segLength < length)
01420                                   {
01421                                       currLen += segLength;
01422                                       currPoint = this.Segments[i].Point;
```

```
01423                                   }
01424                                   else if (segLength > 0)
01425                                   {
01426                                       double pos = (length - currLen) / segLength;
01427                                       return this.Segments[i].GetPointAt(currPoint, pos);
01428                                   }
01429                               }
01430                               break;
01431                           }
01432                       }
01433
01434                       throw new InvalidOperationException("Unexpected code path!");
01435                   }
01436                   else if (length > pathLength)
01437                   {
01438                       double currLength = 0;
01439
01440                       Point currPoint = new Point();
01441                       Point figureStartPoint = new Point();
01442
01443                       for (int i = 0; i < this.Segments.Count - 1; i++)
01444                       {
01445                           switch (this.Segments[i].Type)
01446                           {
01447                               case SegmentType.Move:
01448                                   currPoint = this.Segments[i].Point;
01449                                   figureStartPoint = this.Segments[i].Point;
01450                                   break;
01451                               case SegmentType.Line:
01452                                   if (i > 0)
01453                                   {
01454                                       currLength += this.Segments[i].Measure(currPoint);
01455                                       currPoint = this.Segments[i].Point;
01456                                   }
01457                                   else
01458                                   {
01459                                       currPoint = this.Segments[i].Point;
01460                                       figureStartPoint = this.Segments[i].Point;
01461                                   }
01462                                   break;
01463                               case SegmentType.Arc:
01464                                   if (i > 0)
01465                                   {
01466                                       currLength += this.Segments[i].Measure(currPoint);
01467                                       currPoint = this.Segments[i].Point;
01468                                   }
01469                                   else
01470                                   {
01471                                       ArcSegment seg = (ArcSegment)this.Segments[i];
01472                                       figureStartPoint = new Point(seg.Points[0].X +
    Math.Cos(seg.StartAngle) * seg.Radius, seg.Points[0].Y + Math.Sin(seg.StartAngle) * seg.Radius);
01473                                       currLength += this.Segments[i].Measure(figureStartPoint);
01474                                       currPoint = this.Segments[i].Point;
01475                                   }
01476                                   break;
01477                               case SegmentType.Close:
01478                                   currLength += Math.Sqrt((currPoint.X - figureStartPoint.X) * (currPoint.X
    - figureStartPoint.X) + (currPoint.Y - figureStartPoint.Y) * (currPoint.Y - figureStartPoint.Y));
01479                                   currPoint = figureStartPoint;
01480                                   break;
01481                               case SegmentType.CubicBezier:
01482                                   if (i > 0)
01483                                   {
01484                                       currLength += this.Segments[i].Measure(currPoint);
01485                                       currPoint = this.Segments[i].Point;
01486                                   }
01487                                   else
01488                                   {
01489                                       currPoint = this.Segments[i].Points[0];
01490                                       figureStartPoint = this.Segments[i].Points[0];
01491                                       currLength += this.Segments[i].Measure(currPoint);
01492                                       currPoint = this.Segments[i].Point;
01493                                   }
01494                                   break;
01495                           }
01496                       }
01497
01498                       switch (this.Segments[this.Segments.Count - 1].Type)
01499                       {
01500                           case SegmentType.Arc:
01501                           case SegmentType.CubicBezier:
01502                           case SegmentType.Line:
01503                               {
01504                                   double pos = 1 + (length - pathLength) / this.Segments[this.Segments.Count
    - 1].Measure(currPoint);
01505                                   return this.Segments[this.Segments.Count - 1].GetPointAt(currPoint, pos);
01506                               }
```

```
01507                       case SegmentType.Move:
01508                           return currPoint;
01509                       case SegmentType.Close:
01510                           return this.GetPointAtAbsolute(length - pathLength);
01511                   }
01512
01513                   throw new InvalidOperationException("Unexpected code path!");
01514               }
01515               else
01516               {
01517                   Point currPoint = new Point();
01518                   Point figureStartPoint = new Point();
01519
01520                   for (int i = 0; i < this.Segments.Count; i++)
01521                   {
01522                       switch (this.Segments[i].Type)
01523                       {
01524                           case SegmentType.Move:
01525                               currPoint = this.Segments[i].Point;
01526                               figureStartPoint = this.Segments[i].Point;
01527                               break;
01528                           case SegmentType.Line:
01529                               if (i > 0)
01530                               {
01531                                   double segLength = this.Segments[i].Measure(currPoint);
01532                                   double pos = length / segLength;
01533                                   return this.Segments[i].GetPointAt(currPoint, pos);
01534                               }
01535                               else
01536                               {
01537                                   currPoint = this.Segments[i].Point;
01538                                   figureStartPoint = this.Segments[i].Point;
01539                               }
01540                               break;
01541                           case SegmentType.Arc:
01542                               if (i > 0)
01543                               {
01544                                   double segLength = this.Segments[i].Measure(currPoint);
01545                                   double pos = length / segLength;
01546                                   return this.Segments[i].GetPointAt(currPoint, pos);
01547                               }
01548                               else
01549                               {
01550                                   ArcSegment seg = (ArcSegment)this.Segments[i];
01551                                   figureStartPoint = new Point(seg.Points[0].X +
      Math.Cos(seg.StartAngle) * seg.Radius, seg.Points[0].Y + Math.Sin(seg.StartAngle) * seg.Radius);
01552                                   currPoint = figureStartPoint;
01553
01554                                   double segLength = this.Segments[i].Measure(currPoint);
01555                                   double pos = length / segLength;
01556                                   return this.Segments[i].GetPointAt(currPoint, pos);
01557                               }
01558                           case SegmentType.Close:
01559                               {
01560                                   double segLength = Math.Sqrt((currPoint.X - figureStartPoint.X) *
      (currPoint.X - figureStartPoint.X) + (currPoint.Y - figureStartPoint.Y) * (currPoint.Y -
      figureStartPoint.Y));
01561                                   double pos = length / segLength;
01562                                   return new Point(currPoint.X * (1 - pos) + figureStartPoint.X * pos,
      currPoint.Y * (1 - pos) + figureStartPoint.Y * pos);
01563                               }
01564                           case SegmentType.CubicBezier:
01565                               if (i > 0)
01566                               {
01567                                   double segLength = this.Segments[i].Measure(currPoint);
01568                                   double pos = length / segLength;
01569                                   return this.Segments[i].GetPointAt(currPoint, pos);
01570                               }
01571                               else
01572                               {
01573                                   currPoint = this.Segments[i].Points[0];
01574                                   figureStartPoint = this.Segments[i].Points[0];
01575                                   double segLength = this.Segments[i].Measure(currPoint);
01576                                   double pos = length / segLength;
01577                                   return this.Segments[i].GetPointAt(currPoint, pos);
01578                               }
01579                       }
01580                   }
01581
01582                   throw new InvalidOperationException("Unexpected code path!");
01583               }
01584           }
01585
01586 /// <summary>
01587 /// Gets the tangent to the point at the relative position specified on the <see
      cref="GraphicsPath"/>.
01588 /// </summary>
```

```
01589 /// <param name="position">The position on the <see cref="GraphicsPath"/> (0 is the start of the path,
      1 is the end of the path).</param>
01590 /// <returns>The tangent to the point at the specified position.</returns>
01591         public Point GetTangentAtRelative(double position)
01592         {
01593             return GetTangentAtAbsolute(position * this.MeasureLength());
01594         }
01595 
01596 /// <summary>
01597 /// Gets the tangent to the point at the absolute position specified on the <see
      cref="GraphicsPath"/>.
01598 /// </summary>
01599 /// <param name="length">The distance to the point from the start of the <see
      cref="GraphicsPath"/>.</param>
01600 /// <returns>The tangent to the point at the specified position.</returns>
01601         public Point GetTangentAtAbsolute(double length)
01602         {
01603             double pathLength = this.MeasureLength();
01604 
01605             if (length >= 0 && length <= pathLength)
01606             {
01607                 double currLen = 0;
01608 
01609                 Point currPoint = new Point();
01610                 Point figureStartPoint = new Point();
01611 
01612                 for (int i = 0; i < this.Segments.Count; i++)
01613                 {
01614                     switch (this.Segments[i].Type)
01615                     {
01616                         case SegmentType.Move:
01617                             currPoint = this.Segments[i].Point;
01618                             figureStartPoint = this.Segments[i].Point;
01619                             break;
01620                         case SegmentType.Line:
01621                             if (i > 0)
01622                             {
01623                                 double segLength = this.Segments[i].Measure(currPoint);
01624 
01625                                 if (currLen + segLength < length)
01626                                 {
01627                                     currLen += segLength;
01628                                     currPoint = this.Segments[i].Point;
01629                                 }
01630                                 else
01631                                 {
01632                                     double pos = (length - currLen) / segLength;
01633                                     return this.Segments[i].GetTangentAt(currPoint, pos);
01634                                 }
01635                             }
01636                             else
01637                             {
01638                                 currPoint = this.Segments[i].Point;
01639                                 figureStartPoint = this.Segments[i].Point;
01640                             }
01641                             break;
01642                         case SegmentType.Arc:
01643                             if (i > 0)
01644                             {
01645                                 double segLength = this.Segments[i].Measure(currPoint);
01646 
01647                                 if (currLen + segLength < length)
01648                                 {
01649                                     currLen += segLength;
01650                                     currPoint = this.Segments[i].Point;
01651                                 }
01652                                 else
01653                                 {
01654                                     double pos = (length - currLen) / segLength;
01655                                     return this.Segments[i].GetTangentAt(currPoint, pos);
01656                                 }
01657                             }
01658                             else
01659                             {
01660                                 ArcSegment seg = (ArcSegment)this.Segments[i];
01661                                 figureStartPoint = new Point(seg.Points[0].X +
      Math.Cos(seg.StartAngle) * seg.Radius, seg.Points[0].Y + Math.Sin(seg.StartAngle) * seg.Radius);
01662                                 currPoint = figureStartPoint;
01663 
01664                                 double segLength = this.Segments[i].Measure(currPoint);
01665 
01666                                 if (currLen + segLength < length)
01667                                 {
01668                                     currLen += segLength;
01669                                     currPoint = this.Segments[i].Point;
01670                                 }
01671                                 else
```

```
01672                                   {
01673                                       double pos = (length - currLen) / segLength;
01674                                       return this.Segments[i].GetTangentAt(currPoint, pos);
01675                                   }
01676                               }
01677                               break;
01678                           case SegmentType.Close:
01679                               {
01680                                   double segLength = Math.Sqrt((currPoint.X - figureStartPoint.X) *
      (currPoint.X - figureStartPoint.X) + (currPoint.Y - figureStartPoint.Y) * (currPoint.Y -
      figureStartPoint.Y));
01681
01682                                   if (currLen + segLength < length)
01683                                   {
01684                                       currLen += segLength;
01685                                       currPoint = figureStartPoint;
01686                                   }
01687                                   else
01688                                   {
01689                                       double pos = (length - currLen) / segLength;
01690                                       return new Point(figureStartPoint.X - currPoint.X,
      figureStartPoint.Y - currPoint.Y).Normalize();
01691                                   }
01692                               }
01693                               break;
01694                           case SegmentType.CubicBezier:
01695                               if (i > 0)
01696                               {
01697                                   double segLength = this.Segments[i].Measure(currPoint);
01698
01699                                   if (currLen + segLength < length)
01700                                   {
01701                                       currLen += segLength;
01702                                       currPoint = this.Segments[i].Point;
01703                                   }
01704                                   else
01705                                   {
01706                                       double pos = (length - currLen) / segLength;
01707                                       return this.Segments[i].GetTangentAt(currPoint, pos);
01708                                   }
01709                               }
01710                               else
01711                               {
01712                                   currPoint = this.Segments[i].Points[0];
01713                                   figureStartPoint = this.Segments[i].Points[0];
01714                                   double segLength = this.Segments[i].Measure(currPoint);
01715
01716                                   if (currLen + segLength < length)
01717                                   {
01718                                       currLen += segLength;
01719                                       currPoint = this.Segments[i].Point;
01720                                   }
01721                                   else
01722                                   {
01723                                       double pos = (length - currLen) / segLength;
01724                                       return this.Segments[i].GetTangentAt(currPoint, pos);
01725                                   }
01726                               }
01727                               break;
01728                       }
01729                   }
01730
01731                   throw new InvalidOperationException("Unexpected code path!");
01732               }
01733               else if (length > pathLength)
01734               {
01735                   double currLength = 0;
01736
01737                   Point currPoint = new Point();
01738                   Point figureStartPoint = new Point();
01739
01740                   for (int i = 0; i < this.Segments.Count - 1; i++)
01741                   {
01742                       switch (this.Segments[i].Type)
01743                       {
01744                           case SegmentType.Move:
01745                               currPoint = this.Segments[i].Point;
01746                               figureStartPoint = this.Segments[i].Point;
01747                               break;
01748                           case SegmentType.Line:
01749                               if (i > 0)
01750                               {
01751                                   currLength += this.Segments[i].Measure(currPoint);
01752                                   currPoint = this.Segments[i].Point;
01753                               }
01754                               else
01755                               {
```

```
01756                                   currPoint = this.Segments[i].Point;
01757                                   figureStartPoint = this.Segments[i].Point;
01758                               }
01759                               break;
01760                           case SegmentType.Arc:
01761                               if (i > 0)
01762                               {
01763                                   currLength += this.Segments[i].Measure(currPoint);
01764                                   currPoint = this.Segments[i].Point;
01765                               }
01766                               else
01767                               {
01768                                   ArcSegment seg = (ArcSegment)this.Segments[i];
01769                                   figureStartPoint = new Point(seg.Points[0].X +
      Math.Cos(seg.StartAngle) * seg.Radius, seg.Points[0].Y + Math.Sin(seg.StartAngle) * seg.Radius);
01770                                   currLength += this.Segments[i].Measure(figureStartPoint);
01771                                   currPoint = this.Segments[i].Point;
01772                               }
01773                               break;
01774                           case SegmentType.Close:
01775                               currLength += Math.Sqrt((currPoint.X - figureStartPoint.X) * (currPoint.X
      - figureStartPoint.X) + (currPoint.Y - figureStartPoint.Y) * (currPoint.Y - figureStartPoint.Y));
01776                               currPoint = figureStartPoint;
01777                               break;
01778                           case SegmentType.CubicBezier:
01779                               if (i > 0)
01780                               {
01781                                   currLength += this.Segments[i].Measure(currPoint);
01782                                   currPoint = this.Segments[i].Point;
01783                               }
01784                               else
01785                               {
01786                                   currPoint = this.Segments[i].Points[0];
01787                                   figureStartPoint = this.Segments[i].Points[0];
01788                                   currLength += this.Segments[i].Measure(currPoint);
01789                                   currPoint = this.Segments[i].Point;
01790                               }
01791                               break;
01792                       }
01793                   }
01794
01795                   switch (this.Segments[this.Segments.Count - 1].Type)
01796                   {
01797                       case SegmentType.Arc:
01798                       case SegmentType.CubicBezier:
01799                       case SegmentType.Line:
01800                           {
01801                               double pos = 1 + (length - pathLength) / this.Segments[this.Segments.Count
      - 1].Measure(currPoint);
01802                               return this.Segments[this.Segments.Count - 1].GetTangentAt(currPoint,
      pos);
01803                           }
01804                       case SegmentType.Move:
01805                           return new Point();
01806                       case SegmentType.Close:
01807                           return this.GetTangentAtAbsolute(length - pathLength);
01808                   }
01809
01810                   throw new InvalidOperationException("Unexpected code path!");
01811               }
01812               else
01813               {
01814                   Point currPoint = new Point();
01815                   Point figureStartPoint = new Point();
01816
01817                   for (int i = 0; i < this.Segments.Count; i++)
01818                   {
01819                       switch (this.Segments[i].Type)
01820                       {
01821                           case SegmentType.Move:
01822                               currPoint = this.Segments[i].Point;
01823                               figureStartPoint = this.Segments[i].Point;
01824                               break;
01825                           case SegmentType.Line:
01826                               if (i > 0)
01827                               {
01828                                   double segLength = this.Segments[i].Measure(currPoint);
01829                                   double pos = length / segLength;
01830                                   return this.Segments[i].GetTangentAt(currPoint, pos);
01831                               }
01832                               else
01833                               {
01834                                   currPoint = this.Segments[i].Point;
01835                                   figureStartPoint = this.Segments[i].Point;
01836                               }
01837                               break;
01838                           case SegmentType.Arc:
```

```
01839                               if (i > 0)
01840                               {
01841                                   double segLength = this.Segments[i].Measure(currPoint);
01842                                   double pos = length / segLength;
01843                                   return this.Segments[i].GetTangentAt(currPoint, pos);
01844                               }
01845                               else
01846                               {
01847                                   ArcSegment seg = (ArcSegment)this.Segments[i];
01848                                   figureStartPoint = new Point(seg.Points[0].X +
      Math.Cos(seg.StartAngle) * seg.Radius, seg.Points[0].Y + Math.Sin(seg.StartAngle) * seg.Radius);
01849                                   currPoint = figureStartPoint;
01850 
01851                                   double segLength = this.Segments[i].Measure(currPoint);
01852                                   double pos = length / segLength;
01853                                   return this.Segments[i].GetTangentAt(currPoint, pos);
01854                               }
01855                           case SegmentType.Close:
01856                               {
01857                                   double segLength = Math.Sqrt((currPoint.X - figureStartPoint.X) *
      (currPoint.X - figureStartPoint.X) + (currPoint.Y - figureStartPoint.Y) * (currPoint.Y -
      figureStartPoint.Y));
01858                                   double pos = length / segLength;
01859                                   return new Point(figureStartPoint.X - currPoint.X, figureStartPoint.Y
      - currPoint.Y).Normalize();
01860                               }
01861                           case SegmentType.CubicBezier:
01862                               if (i > 0)
01863                               {
01864                                   double segLength = this.Segments[i].Measure(currPoint);
01865                                   double pos = length / segLength;
01866                                   return this.Segments[i].GetTangentAt(currPoint, pos);
01867                               }
01868                               else
01869                               {
01870                                   currPoint = this.Segments[i].Points[0];
01871                                   figureStartPoint = this.Segments[i].Points[0];
01872                                   double segLength = this.Segments[i].Measure(currPoint);
01873                                   double pos = length / segLength;
01874                                   return this.Segments[i].GetTangentAt(currPoint, pos);
01875                               }
01876                       }
01877                   }
01878 
01879                   throw new InvalidOperationException("Unexpected code path!");
01880               }
01881           }
01882 
01883 /// <summary>
01884 /// Gets the normal to the point at the absolute position specified on the <see cref="GraphicsPath"/>.
01885 /// </summary>
01886 /// <param name="length">The distance to the point from the start of the <see
      cref="GraphicsPath"/>.</param>
01887 /// <returns>The normal to the point at the specified position.</returns>
01888         public Point GetNormalAtAbsolute(double length)
01889         {
01890             Point tangent = this.GetTangentAtAbsolute(length);
01891             return new Point(-tangent.Y, tangent.X);
01892         }
01893 
01894 /// <summary>
01895 /// Gets the normal to the point at the relative position specified on the <see cref="GraphicsPath"/>.
01896 /// </summary>
01897 /// <param name="position">The position on the <see cref="GraphicsPath"/> (0 is the start of the path,
      1 is the end of the path).</param>
01898 /// <returns>The normal to the point at the specified position.</returns>
01899         public Point GetNormalAtRelative(double position)
01900         {
01901             Point tangent = this.GetTangentAtRelative(position);
01902             return new Point(-tangent.Y, tangent.X);
01903         }
01904 
01905 /// <summary>
01906 /// Linearises a <see cref="GraphicsPath"/>, replacing curve segments with series of line segments
      that approximate them.
01907 /// </summary>
01908 /// <param name="resolution">The absolute length between successive samples in curve segments.</param>
01909 /// <returns>A <see cref="GraphicsPath"/> composed only of linear segments that approximates the
      current <see cref="GraphicsPath"/>.</returns>
01910         [Pure]
01911         public GraphicsPath Linearise(double resolution)
01912         {
01913             if (!(resolution > 0))
01914             {
01915                 throw new ArgumentOutOfRangeException(nameof(resolution), resolution, "The resolution
      must be greater than 0!");
01916             }
```

```
01917
01918             GraphicsPath tbr = new GraphicsPath();
01919
01920             Point?[] previousPoints = new Point?[this.Segments.Count];
01921
01922             previousPoints[0] = null;
01923
01924             for (int i = 0; i < this.Segments.Count - 1; i++)
01925             {
01926                 if (this.Segments[i].Type != SegmentType.Close)
01927                 {
01928                     previousPoints[i + 1] = this.Segments[i].Point;
01929                 }
01930                 else
01931                 {
01932                     previousPoints[i + 1] = previousPoints[i];
01933                 }
01934             }
01935
01936             Segment[][] linearisedSegments = new Segment[this.Segments.Count][];
01937
01938             System.Threading.Tasks.Parallel.For(0, this.Segments.Count, i =>
01939             {
01940                 linearisedSegments[i] = this.Segments[i].Linearise(previousPoints[i],
      resolution).ToArray();
01941             });
01942
01943             int total = 0;
01944
01945             for (int i = 0; i < linearisedSegments.Length; i++)
01946             {
01947                 total += linearisedSegments[i].Length;
01948             }
01949
01950             tbr.Segments.Capacity = total;
01951
01952             for (int i = 0; i < linearisedSegments.Length; i++)
01953             {
01954                 tbr.Segments.AddRange(linearisedSegments[i]);
01955             }
01956
01957             return tbr;
01958         }
01959
01960 /// <summary>
01961 /// Discretises a <see cref="GraphicsPath"/>, replacing curve segments with series of line segments
      that approximate them and ensuring that all line segments are shorter than the specified <paramref
      name="resolution"/>.
01962 /// </summary>
01963 /// <param name="resolution">The maximum length (in absolute units) of line segments in the resulting
      <see cref="GraphicsPath"/>.</param>
01964 /// <returns>A <see cref="GraphicsPath"/> composed only of linear segments that are shorter than
      <paramref name="resolution"/> and approximate the current <see cref="GraphicsPath"/>.</returns>
01965         [Pure]
01966         public GraphicsPath Discretise(double resolution)
01967         {
01968             if (!(resolution > 0))
01969             {
01970                 throw new ArgumentOutOfRangeException(nameof(resolution), resolution, "The resolution
      must be greater than 0!");
01971             }
01972
01973             GraphicsPath tbr = new GraphicsPath();
01974
01975             Point?  previousPoint = null;
01976
01977             foreach (Segment seg in this.Segments)
01978             {
01979                 tbr.Segments.AddRange(seg.Linearise(previousPoint, resolution));
01980
01981                 if (seg.Type != SegmentType.Close)
01982                 {
01983                     previousPoint = seg.Point;
01984                 }
01985             }
01986
01987             return Graphics.ReduceMaximumLength(tbr, resolution);
01988         }
01989
01990
01991 /// <summary>
01992 /// Flattens a <see cref="GraphicsPath"/>, replacing curve segments with series of line segments that
      approximate them, ensuring the specified maximum deviation from the original path.
01993 /// </summary>
01994 /// <param name="flatness">The maximum deviation from the original path.</param>
01995 /// <returns>A <see cref="GraphicsPath"/> composed only of linear segments that approximates the
      current <see cref="GraphicsPath"/>.</returns>
```

```
01996         [Pure]
01997         public GraphicsPath Flatten(double flatness)
01998         {
01999             if (!(flatness > 0))
02000             {
02001                 throw new ArgumentOutOfRangeException(nameof(flatness), flatness, "The flatness must
      be greater than 0!");
02002             }
02003 
02004             GraphicsPath tbr = new GraphicsPath();
02005 
02006             Point?[] previousPoints = new Point?[this.Segments.Count];
02007 
02008             previousPoints[0] = null;
02009 
02010             for (int i = 0; i < this.Segments.Count - 1; i++)
02011             {
02012                 if (this.Segments[i].Type != SegmentType.Close)
02013                 {
02014                     previousPoints[i + 1] = this.Segments[i].Point;
02015                 }
02016                 else
02017                 {
02018                     previousPoints[i + 1] = previousPoints[i];
02019                 }
02020             }
02021 
02022             Segment[][] flattenedSegments = new Segment[this.Segments.Count][];
02023 
02024             System.Threading.Tasks.Parallel.For(0, this.Segments.Count, i =>
02025             {
02026                 flattenedSegments[i] = this.Segments[i].Flatten(previousPoints[i],
      flatness).ToArray();
02027             });
02028 
02029             int total = 0;
02030 
02031             for (int i = 0; i < flattenedSegments.Length; i++)
02032             {
02033                 total += flattenedSegments[i].Length;
02034             }
02035 
02036             tbr.Segments.Capacity = total;
02037 
02038             for (int i = 0; i < flattenedSegments.Length; i++)
02039             {
02040                 tbr.Segments.AddRange(flattenedSegments[i]);
02041             }
02042 
02043             return tbr;
02044         }
02045 
02046 /// <summary>
02047 /// Gets a collection of the end points of all the segments in the <see cref="GraphicsPath"/>, divided
      by figure.
02048 /// </summary>
02049 /// <returns>A collection of the end points of all the segments in the <see cref="GraphicsPath"/>,
      divided by figure.</returns>
02050         public IEnumerable<List<Point» GetPoints()
02051         {
02052             Point startPoint = new Point();
02053 
02054             List<Point> currFigure = null;
02055             bool returned = true;
02056 
02057             foreach (Segment seg in this.Segments)
02058             {
02059                 if (seg.Type != SegmentType.Close)
02060                 {
02061                     Point currPoint = seg.Point;
02062                     if (seg.Type == SegmentType.Move)
02063                     {
02064                         if (!returned)
02065                         {
02066                             yield return currFigure;
02067                         }
02068 
02069                         startPoint = currPoint;
02070                         currFigure = new List<Point>();
02071                         returned = false;
02072                     }
02073                     currFigure.Add(currPoint);
02074                 }
02075                 else
02076                 {
02077                     currFigure.Add(startPoint);
02078                     yield return currFigure;
```

```
02079                     returned = true;
02080                 }
02081             }
02082
02083             if (!returned)
02084             {
02085                 yield return currFigure;
02086             }
02087         }
02088
02089 /// <summary>
02090 /// Gets a collection of all the figures in the <see cref="GraphicsPath"/>, returned as individual
      <see cref="GraphicsPath"/>s.
02091 /// </summary>
02092 /// <returns>A collection of all the figures in the <see cref="GraphicsPath"/>, returned as individual
      <see cref="GraphicsPath"/>s.</returns>
02093         public IEnumerable<GraphicsPath> GetFigures()
02094         {
02095             GraphicsPath currFigure = null;
02096             bool returned = true;
02097
02098             foreach (Segment seg in this.Segments)
02099             {
02100                 if (seg.Type != SegmentType.Close)
02101                 {
02102                     Point currPoint = seg.Point;
02103                     if (seg.Type == SegmentType.Move)
02104                     {
02105                         if (!returned)
02106                         {
02107                             yield return currFigure;
02108                         }
02109
02110                         currFigure = new GraphicsPath();
02111                         returned = false;
02112                     }
02113
02114                     currFigure.Segments.Add(seg);
02115                 }
02116                 else
02117                 {
02118                     currFigure.Close();
02119                     yield return currFigure;
02120                     returned = true;
02121                 }
02122             }
02123
02124             if (!returned)
02125             {
02126                 yield return currFigure;
02127             }
02128         }
02129
02130
02131 /// <summary>
02132 /// Gets a collection of the tangents at the end point of the segments in which the <see
      cref="GraphicsPath"/> would be linearised, divided by figure.
02133 /// </summary>
02134 /// <param name="resolution">The absolute length between successive samples in curve segments.</param>
02135 /// <returns>A collection of the tangents at the end point of the segments in which the <see
      cref="GraphicsPath"/> would be linearised, divided by figure.</returns>
02136         public IEnumerable<List<Point» GetLinearisationPointsNormals(double resolution)
02137         {
02138             if (!(resolution > 0))
02139             {
02140                 throw new ArgumentOutOfRangeException(nameof(resolution), resolution, "The resolution
      must be greater than 0!");
02141             }
02142
02143             Point previousPoint = new Point();
02144             Point startPoint = new Point();
02145
02146             List<Point> currFigure = null;
02147             bool returned = true;
02148
02149             for (int i = 0; i < this.Segments.Count; i++)
02150             {
02151                 Segment seg = this.Segments[i];
02152
02153                 if (seg.Type != SegmentType.Close)
02154                 {
02155                     Point currPoint = seg.Point;
02156                     if (seg.Type == SegmentType.Move)
02157                     {
02158                         if (!returned)
02159                         {
02160                             yield return currFigure;
```

```
02161                         }
02162
02163                         startPoint = currPoint;
02164                         currFigure = new List<Point>();
02165                         returned = false;
02166
02167                         if (i < this.Segments.Count - 1 && this.Segments[i + 1].Type !=
      SegmentType.Move)
02168                         {
02169                             Point tangent = this.Segments[i + 1].GetTangentAt(seg.Point, 0);
02170
02171                             currFigure.Add(new Point(-tangent.Y, tangent.X));
02172                         }
02173                         else
02174                         {
02175                             currFigure.Add(new Point());
02176                         }
02177                     }
02178                     else
02179                     {
02180                         foreach (Point tangent in seg.GetLinearisationTangents(previousPoint,
      resolution))
02181                         {
02182                             currFigure.Add(new Point(-tangent.Y, tangent.X));
02183                         }
02184                     }
02185
02186                     previousPoint = currPoint;
02187                 }
02188                 else
02189                 {
02190                     Point normal;
02191
02192                     if (!startPoint.IsEqual(previousPoint, 1e-4))
02193                     {
02194                         Point tangent = new Point(startPoint.X - previousPoint.X, startPoint.Y -
      previousPoint.Y).Normalize();
02195                         normal = new Point(-tangent.Y, tangent.X);
02196                     }
02197                     else
02198                     {
02199                         normal = currFigure[currFigure.Count - 1];
02200                     }
02201
02202                     currFigure.Add(normal);
02203                     currFigure[0] = new Point((currFigure[1].X + normal.X) * 0.5, (currFigure[1].Y +
      normal.Y) * 0.5).Normalize();
02204
02205                     yield return currFigure;
02206                     returned = true;
02207                 }
02208             }
02209
02210             if (!returned)
02211             {
02212                 yield return currFigure;
02213             }
02214
02215         }
02216
02217
02218         private enum VertexType
02219         {
02220             Start, End, Regular, Split, Merge
02221         };
02222
02223 /// <summary>
02224 /// Divides a <see cref="GraphicsPath"/> into triangles.
02225 /// </summary>
02226 /// <param name="resolution">The resolution that will be used to linearise curve segments in the <see
      cref="GraphicsPath"/>.</param>
02227 /// <param name="clockwise">If this is <see langword="true"/>, the triangles will have their vertices
      in a clockwise order, otherwise they will be in anticlockwise order.</param>
02228 /// <returns>A collection of distinct <see cref="GraphicsPath"/>s, each representing one
      triangle.</returns>
02229         [Pure]
02230         public IEnumerable<GraphicsPath> Triangulate(double resolution, bool clockwise)
02231         {
02232             double shiftAmount = 0.01 * resolution;
02233
02234             if (!(resolution > 0))
02235             {
02236                 throw new ArgumentOutOfRangeException(nameof(resolution), resolution, "The resolution
      must be greater than 0!");
02237             }
02238
02239             GraphicsPath linearisedPath = this.Linearise(resolution);
```

```
02240
02241             List<Point> vertices = new List<Point>();
02242             List<List<int» vertexEdges = new List<List<int»();
02243             List<(int, int)> edges = new List<(int, int)>();
02244             int lastStartingPoint = -1;
02245             int lastSegmentEnd = -1;
02246             double area = 0;
02247
02248             foreach (Segment seg in linearisedPath.Segments)
02249             {
02250                 if (seg is MoveSegment)
02251                 {
02252                     vertices.Add(seg.Point);
02253                     vertexEdges.Add(new List<int>(2));
02254                     lastStartingPoint = vertices.Count - 1;
02255                     lastSegmentEnd = vertices.Count - 1;
02256                 }
02257                 else if (seg is LineSegment)
02258                 {
02259                     if (!vertices[lastSegmentEnd].IsEqual(seg.Point, 1e-4))
02260                     {
02261                         vertices.Add(seg.Point);
02262                         vertexEdges.Add(new List<int>(2));
02263                         edges.Add((lastSegmentEnd, vertices.Count - 1));
02264                         area += (seg.Point.X - vertices[lastSegmentEnd].X) * (seg.Point.Y +
    vertices[lastSegmentEnd].Y);
02265                         vertexEdges[lastSegmentEnd].Add(edges.Count - 1);
02266                         vertexEdges[vertices.Count - 1].Add(edges.Count - 1);
02267                         lastSegmentEnd = vertices.Count - 1;
02268                     }
02269                 }
02270                 else if (seg is CloseSegment)
02271                 {
02272                     if (!vertices[lastSegmentEnd].IsEqual(vertices[lastStartingPoint], 1e-4))
02273                     {
02274                         edges.Add((lastSegmentEnd, lastStartingPoint));
02275                         area += (vertices[lastStartingPoint].X - vertices[lastSegmentEnd].X) *
    (vertices[lastStartingPoint].Y + vertices[lastSegmentEnd].Y);
02276                         vertexEdges[lastSegmentEnd].Add(edges.Count - 1);
02277                         vertexEdges[lastStartingPoint].Add(edges.Count - 1);
02278                     }
02279                     else
02280                     {
02281                         vertices.RemoveAt(lastSegmentEnd);
02282                         vertexEdges.RemoveAt(lastSegmentEnd);
02283
02284                         for (int i = 0; i < edges.Count; i++)
02285                         {
02286                             if (edges[i].Item1 == lastSegmentEnd)
02287                             {
02288                                 edges[i] = (lastStartingPoint, edges[i].Item2);
02289                                 vertexEdges[lastStartingPoint].Add(i);
02290                             }
02291                             else if (edges[i].Item2 == lastSegmentEnd)
02292                             {
02293                                 edges[i] = (edges[i].Item1, lastStartingPoint);
02294                                 vertexEdges[lastStartingPoint].Add(i);
02295                             }
02296                         }
02297                     }
02298
02299                     lastStartingPoint = -1;
02300                     lastSegmentEnd = -1;
02301                 }
02302             }
02303
02304             if (vertices.Count < 3)
02305             {
02306                 yield break;
02307             }
02308
02309             bool isAntiClockwise = area > 0;
02310
02311             int compareVertices(Point a, Point b)
02312             {
02313                 if (a.Y - b.Y != 0)
02314                 {
02315                     return Math.Sign(a.Y - b.Y);
02316                 }
02317                 else
02318                 {
02319                     return Math.Sign(a.X - b.X);
02320                 }
02321             }
02322
02323             Dictionary<double, int> yCoordinates = new Dictionary<double, int>();
02324             Dictionary<double, int> yShiftCount = new Dictionary<double, int>();
```

```
02325
02326             foreach (Point pt in vertices)
02327             {
02328                 if (yCoordinates.ContainsKey(pt.Y))
02329                 {
02330                     yCoordinates[pt.Y]++;
02331                 }
02332                 else
02333                 {
02334                     yCoordinates[pt.Y] = 1;
02335                     yShiftCount[pt.Y] = 0;
02336                 }
02337             }
02338
02339             HashSet<double> yS = new HashSet<double>(from el in yCoordinates select el.Key);
02340
02341             for (int i = 0; i < vertices.Count; i++)
02342             {
02343                 if (yCoordinates[vertices[i].Y] > 1)
02344                 {
02345                     int shiftCount = yShiftCount[vertices[i].Y];
02346
02347                     double targetCoordinate;
02348
02349                     do
02350                     {
02351                         shiftCount++;
02352
02353                         targetCoordinate = vertices[i].Y + (2 * (shiftCount % 2) - 1) * (1 -
      Math.Pow(0.5, (shiftCount - 1) / 2 + 1)) * shiftAmount;
02354                     }
02355                     while (yS.Contains(targetCoordinate));
02356
02357                     yS.Add(targetCoordinate);
02358                     yCoordinates[vertices[i].Y]--;
02359                     yShiftCount[vertices[i].Y] = shiftCount;
02360                     vertices[i] = new Point(vertices[i].X, targetCoordinate);
02361                 }
02362
02363             }
02364
02365             Queue<int> sortedVertices = new Queue<int>(Enumerable.Range(0, vertices.Count).OrderBy(i
      => vertices[i], Comparer<Point>.Create(compareVertices)));
02366
02367             VertexType[] vertexTypes = new VertexType[vertices.Count];
02368
02369             List<(int, int)> exploredEdges = new List<(int, int)>();
02370             List<int> helpers = new List<int>();
02371             List<(int, int)> diagonals = new List<(int, int)>();
02372             int[] nexts = new int[vertices.Count];
02373             int[] prevs = new int[vertices.Count];
02374
02375             while (sortedVertices.Count > 0)
02376             {
02377                 int vertex = sortedVertices.Dequeue();
02378
02379                 Point pt = vertices[vertex];
02380
02381                 (int, int) edge1 = edges[vertexEdges[vertex][0]];
02382                 (int, int) edge2 = edges[vertexEdges[vertex][1]];
02383
02384                 int neighbour1 = edge1.Item1 != vertex ?  edge1.Item1 :  edge1.Item2;
02385                 int neighbour2 = edge2.Item1 != vertex ?  edge2.Item1 :  edge2.Item2;
02386
02387                 int minNeighbour = Math.Min(neighbour1, neighbour2);
02388                 int maxNeighbour = Math.Max(neighbour1, neighbour2);
02389
02390                 int prev, next;
02391
02392                 if (vertex - minNeighbour == 1 && maxNeighbour - vertex == 1)
02393                 {
02394                     prev = minNeighbour;
02395                     next = maxNeighbour;
02396                 }
02397                 else if ((minNeighbour - vertex == 1 && maxNeighbour - vertex > 1) || vertex -
      maxNeighbour == 1 && vertex - minNeighbour > 1)
02398                 {
02399                     prev = maxNeighbour;
02400                     next = minNeighbour;
02401                 }
02402                 else
02403                 {
02404                     throw new InvalidOperationException("Could not make sense of the ordering of the
      vertices!");
02405                 }
02406
02407                 nexts[vertex] = next;
```

```
02408                 prevs[vertex] = prev;
02409
02410                 Point prevPoint = vertices[prev];
02411                 Point nextPoint = vertices[next];
02412
02413                 double angle = Math.Atan2(prevPoint.Y - pt.Y, prevPoint.X - pt.X) -
      Math.Atan2(nextPoint.Y - pt.Y, nextPoint.X - pt.X);
02414
02415                 if (angle < 0)
02416                 {
02417                     angle += 2 * Math.PI;
02418                 }
02419
02420                 VertexType vertexType;
02421
02422                 if (prevPoint.Y >= pt.Y && nextPoint.Y >= pt.Y && angle < Math.PI)
02423                 {
02424                     vertexType = VertexType.Start;
02425                 }
02426                 else if (prevPoint.Y >= pt.Y && nextPoint.Y >= pt.Y && angle > Math.PI)
02427                 {
02428                     vertexType = VertexType.Split;
02429                 }
02430                 else if (prevPoint.Y <= pt.Y && nextPoint.Y <= pt.Y && angle < Math.PI)
02431                 {
02432                     vertexType = VertexType.End;
02433                 }
02434                 else if (prevPoint.Y <= pt.Y && nextPoint.Y <= pt.Y && angle > Math.PI)
02435                 {
02436                     vertexType = VertexType.Merge;
02437                 }
02438                 else
02439                 {
02440                     vertexType = VertexType.Regular;
02441                 }
02442
02443                 vertexTypes[vertex] = vertexType;
02444
02445                 //gpr.FillText(vertices[vertex], vertex.ToString(), new Font(new
      FontFamily(FontFamily.StandardFontFamilies.Helvetica), 4), Colours.Orange);
02446
02447                 if (vertexType == VertexType.Start)
02448                 {
02449                     exploredEdges.Add((prev, vertex));
02450                     helpers.Add(vertex);
02451
02452                     //gpr.StrokeRectangle(pt.X - 1, pt.Y - 1, 2, 2, Colours.Green, 0.25);
02453                 }
02454                 else if (vertexType == VertexType.End)
02455                 {
02456                     int eiM1 = -1;
02457
02458                     for (int i = exploredEdges.Count - 1; i >= 0; i--)
02459                     {
02460                         if (exploredEdges[i].Item1 == vertex && exploredEdges[i].Item2 == next)
02461                         {
02462                             eiM1 = i;
02463                             break;
02464                         }
02465                     }
02466
02467                     if (eiM1 >= 0)
02468                     {
02469                         if (vertexTypes[helpers[eiM1]] == VertexType.Merge)
02470                         {
02471                             diagonals.Add((helpers[eiM1], vertex));
02472                         }
02473
02474                         exploredEdges.RemoveAt(eiM1);
02475                         helpers.RemoveAt(eiM1);
02476                     }
02477                 }
02478                 else if (vertexType == VertexType.Split)
02479                 {
02480                     (int, int) ej = (-1, -1);
02481                     int ejIndex = -1;
02482
02483                     double xJ = double.MinValue;
02484
02485                     for (int i = 0; i < exploredEdges.Count; i++)
02486                     {
02487                         if ((vertices[exploredEdges[i].Item1].Y <= pt.Y &&
      vertices[exploredEdges[i].Item2].Y >= pt.Y) || (vertices[exploredEdges[i].Item1].Y >= pt.Y &&
      vertices[exploredEdges[i].Item2].Y <= pt.Y))
02488                         {
02489                             double dy = pt.Y - vertices[exploredEdges[i].Item1].Y;
02490                             double dx = dy * (vertices[exploredEdges[i].Item2].X -
```

```
      vertices[exploredEdges[i].Item1].X) / (vertices[exploredEdges[i].Item2].Y -
      vertices[exploredEdges[i].Item1].Y);
02491
02492                             double x = dx + vertices[exploredEdges[i].Item1].X;
02493
02494                             if (x < pt.X && x >= xJ)
02495                             {
02496                                 xJ = x;
02497                                 ej = exploredEdges[i];
02498                                 ejIndex = i;
02499                             }
02500                         }
02501                     }
02502
02503                     if (ejIndex >= 0)
02504                     {
02505                         diagonals.Add((helpers[ejIndex], vertex));
02506
02507                         helpers[ejIndex] = vertex;
02508
02509                         exploredEdges.Add((prev, vertex));
02510                         helpers.Add(vertex);
02511                     }
02512                 }
02513                 else if (vertexType == VertexType.Merge)
02514                 {
02515                     int eiM1 = -1;
02516
02517                     for (int i = exploredEdges.Count - 1; i >= 0; i--)
02518                     {
02519                         if (exploredEdges[i].Item1 == vertex && exploredEdges[i].Item2 == next)
02520                         {
02521                             eiM1 = i;
02522                             break;
02523                         }
02524                     }
02525
02526                     if (eiM1 >= 0)
02527                     {
02528                         if (vertexTypes[helpers[eiM1]] == VertexType.Merge)
02529                         {
02530                             diagonals.Add((helpers[eiM1], vertex));
02531                         }
02532
02533                         exploredEdges.RemoveAt(eiM1);
02534                         helpers.RemoveAt(eiM1);
02535                     }
02536
02537                     (int, int) ej = (-1, -1);
02538                     int ejIndex = -1;
02539
02540                     double xJ = double.MinValue;
02541
02542                     for (int i = 0; i < exploredEdges.Count; i++)
02543                     {
02544                         if ((vertices[exploredEdges[i].Item1].Y <= pt.Y &&
      vertices[exploredEdges[i].Item2].Y >= pt.Y) || (vertices[exploredEdges[i].Item1].Y >= pt.Y &&
      vertices[exploredEdges[i].Item2].Y <= pt.Y))
02545                         {
02546                             double dy = pt.Y - vertices[exploredEdges[i].Item1].Y;
02547                             double dx = dy * (vertices[exploredEdges[i].Item2].X -
      vertices[exploredEdges[i].Item1].X) / (vertices[exploredEdges[i].Item2].Y -
      vertices[exploredEdges[i].Item1].Y);
02548
02549                             double x = dx + vertices[exploredEdges[i].Item1].X;
02550
02551                             if (x < pt.X && x >= xJ)
02552                             {
02553                                 xJ = x;
02554                                 ej = exploredEdges[i];
02555                                 ejIndex = i;
02556                             }
02557                         }
02558                     }
02559
02560                     if (ejIndex >= 0)
02561                     {
02562                         if (vertexTypes[helpers[ejIndex]] == VertexType.Merge)
02563                         {
02564                             diagonals.Add((helpers[ejIndex], vertex));
02565                         }
02566
02567                         helpers[ejIndex] = vertex;
02568                     }
02569                 }
02570                 else if (vertexType == VertexType.Regular)
02571                 {
```

```
02572                     if ((isAntiClockwise && (prevPoint.Y < pt.Y || pt.Y < nextPoint.Y)) ||
      (!isAntiClockwise && (prevPoint.Y > pt.Y || pt.Y > nextPoint.Y)))
02573                     {
02574                         int eiM1 = -1;
02575 
02576                         for (int i = exploredEdges.Count - 1; i >= 0; i--)
02577                         {
02578                             if (exploredEdges[i].Item1 == vertex && exploredEdges[i].Item2 == next)
02579                             {
02580                                 eiM1 = i;
02581                                 break;
02582                             }
02583                         }
02584 
02585                         if (eiM1 >= 0)
02586                         {
02587                             if (vertexTypes[helpers[eiM1]] == VertexType.Merge)
02588                             {
02589                                 diagonals.Add((helpers[eiM1], vertex));
02590                             }
02591 
02592                             exploredEdges.RemoveAt(eiM1);
02593                             helpers.RemoveAt(eiM1);
02594                         }
02595 
02596                         exploredEdges.Add((prev, vertex));
02597                         helpers.Add(vertex);
02598                     }
02599                     else
02600                     {
02601                         (int, int) ej = (-1, -1);
02602                         int ejIndex = -1;
02603 
02604                         double xJ = double.MinValue;
02605 
02606                         for (int i = 0; i < exploredEdges.Count; i++)
02607                         {
02608                             if ((vertices[exploredEdges[i].Item1].Y <= pt.Y &&
      vertices[exploredEdges[i].Item2].Y >= pt.Y) || (vertices[exploredEdges[i].Item1].Y >= pt.Y &&
      vertices[exploredEdges[i].Item2].Y <= pt.Y))
02609                             {
02610                                 double dy = pt.Y - vertices[exploredEdges[i].Item1].Y;
02611                                 double dx = dy * (vertices[exploredEdges[i].Item2].X -
      vertices[exploredEdges[i].Item1].X) / (vertices[exploredEdges[i].Item2].Y -
      vertices[exploredEdges[i].Item1].Y);
02612 
02613                                 double x = dx + vertices[exploredEdges[i].Item1].X;
02614 
02615                                 if (x < pt.X && x >= xJ)
02616                                 {
02617                                     xJ = x;
02618                                     ej = exploredEdges[i];
02619                                     ejIndex = i;
02620                                 }
02621                             }
02622                         }
02623 
02624                         if (ejIndex >= 0)
02625                         {
02626                             if (vertexTypes[helpers[ejIndex]] == VertexType.Merge)
02627                             {
02628                                 diagonals.Add((helpers[ejIndex], vertex));
02629                             }
02630 
02631                             helpers[ejIndex] = vertex;
02632                         }
02633                     }
02634                 }
02635             }
02636 
02637             for (int i = diagonals.Count - 1; i >= 0; i--)
02638             {
02639                 for (int j = 0; j < edges.Count; j++)
02640                 {
02641                     if (CompareEdges(diagonals[i], edges[j]))
02642                     {
02643                         diagonals.RemoveAt(i);
02644                         break;
02645                     }
02646                 }
02647             }
02648 
02649             List<List<(int, int)» polygons = SplitPolygons(edges, diagonals, vertices, isAntiClockwise
      ?  prevs :  nexts);
02650 
02651             int[] directions = new int[vertices.Count];
02652 
```

```
02653             int ind = 0;
02654
02655             foreach (List<(int, int)> polygon in polygons)
02656             {
02657                 foreach (GraphicsPath pth in TriangulateMonotone(vertices, polygon, directions,
      clockwise ?  -1 :  1))
02658                 {
02659                     yield return pth;
02660                 }
02661
02662                 ind++;
02663             }
02664         }
02665
02666         private static bool CompareEdges((int, int) edge1, (int, int) edge2)
02667         {
02668             return (edge1.Item1 == edge2.Item1 && edge1.Item2 == edge2.Item2) || (edge1.Item1 ==
      edge2.Item2 && edge1.Item2 == edge2.Item1);
02669         }
02670
02671         private static List<List<(int, int)» SplitPolygons(List<(int, int)> edges, List<(int, int)>
      diagonals, List<Point> vertices, int[] nexts)
02672         {
02673             List<List<(int, int)» polygons = new List<List<(int, int)»();
02674
02675             List<int>[] outPaths = new List<int>[vertices.Count];
02676
02677             for (int i = 0; i < edges.Count; i++)
02678             {
02679                 if (outPaths[edges[i].Item1] == null)
02680                 {
02681                     outPaths[edges[i].Item1] = new List<int>();
02682                 }
02683
02684                 if (outPaths[edges[i].Item2] == null)
02685                 {
02686                     outPaths[edges[i].Item2] = new List<int>();
02687                 }
02688
02689                 outPaths[edges[i].Item1].Add(edges[i].Item2);
02690                 outPaths[edges[i].Item2].Add(edges[i].Item1);
02691             }
02692
02693             for (int i = 0; i < diagonals.Count; i++)
02694             {
02695                 if (outPaths[diagonals[i].Item1] == null)
02696                 {
02697                     outPaths[diagonals[i].Item1] = new List<int>();
02698                 }
02699
02700                 if (outPaths[diagonals[i].Item2] == null)
02701                 {
02702                     outPaths[diagonals[i].Item2] = new List<int>();
02703                 }
02704
02705                 outPaths[diagonals[i].Item1].Add(diagonals[i].Item2);
02706                 outPaths[diagonals[i].Item1].Add(diagonals[i].Item2);
02707                 outPaths[diagonals[i].Item2].Add(diagonals[i].Item1);
02708                 outPaths[diagonals[i].Item2].Add(diagonals[i].Item1);
02709             }
02710
02711             List<int> activeVertices = (from el in Enumerable.Range(0, outPaths.Length) where
      outPaths[el] != null && outPaths[el].Count > 0 select el).ToList();
02712
02713             int[] newNexts = new int[nexts.Length];
02714
02715             for (int i = 0; i < nexts.Length; i++)
02716             {
02717                 if (activeVertices.Contains(i))
02718                 {
02719                     int currNext = nexts[i];
02720                     while (!outPaths[i].Contains(currNext))
02721                     {
02722                         currNext = nexts[currNext];
02723                     }
02724                     newNexts[i] = currNext;
02725                 }
02726                 else
02727                 {
02728                     newNexts[i] = -1;
02729                 }
02730             }
02731
02732             while (activeVertices.Count > 0)
02733             {
02734                 List<(int, int)> polygon = new List<(int, int)>();
02735
```

```
02736                 int startPoint = activeVertices.Min();
02737
02738                 if (!outPaths[startPoint].Contains(newNexts[startPoint]))
02739                 {
02740                     throw new InvalidOperationException("Missing edge!");
02741                 }
02742
02743                 int prevVertex = startPoint;
02744                 int currVertex = newNexts[startPoint];
02745
02746                 outPaths[startPoint].Remove(currVertex);
02747                 outPaths[currVertex].Remove(startPoint);
02748                 polygon.Add((startPoint, currVertex));
02749
02750                 while (currVertex != startPoint)
02751                 {
02752                     Point currPoint = vertices[currVertex];
02753                     Point prevPoint = vertices[prevVertex];
02754
02755                     double angleIncoming = Math.Atan2(prevPoint.Y - currPoint.Y, prevPoint.X -
      currPoint.X);
02756                     if (angleIncoming < 0)
02757                     {
02758                         angleIncoming += 2 * Math.PI;
02759                     }
02760
02761                     double maxAngle = double.MinValue;
02762                     int candidateVertex = -1;
02763
02764                     for (int i = 0; i < outPaths[currVertex].Count; i++)
02765                     {
02766                         double angleI = Math.Atan2(vertices[outPaths[currVertex][i]].Y - currPoint.Y,
      vertices[outPaths[currVertex][i]].X - currPoint.X);
02767                         if (angleI < 0)
02768                         {
02769                             angleI += 2 * Math.PI;
02770                         }
02771                         angleI -= angleIncoming;
02772                         if (angleI < 0)
02773                         {
02774                             angleI += 2 * Math.PI;
02775                         }
02776
02777                         if (angleI > maxAngle)
02778                         {
02779                             candidateVertex = outPaths[currVertex][i];
02780                             maxAngle = angleI;
02781                         }
02782                     }
02783
02784                     outPaths[currVertex].Remove(candidateVertex);
02785                     outPaths[candidateVertex].Remove(currVertex);
02786                     polygon.Add((currVertex, candidateVertex));
02787
02788                     prevVertex = currVertex;
02789                     currVertex = candidateVertex;
02790                 }
02791
02792                 polygons.Add(polygon);
02793                 activeVertices = (from el in Enumerable.Range(0, outPaths.Length) where outPaths[el]
      != null && outPaths[el].Count > 0 select el).ToList();
02794
02795                 if (activeVertices.Contains(currVertex))
02796                 {
02797                     int currNext = newNexts[currVertex];
02798                     while (!outPaths[currVertex].Contains(currNext))
02799                     {
02800                         currNext = newNexts[currNext];
02801                     }
02802                     newNexts[currVertex] = currNext;
02803                 }
02804
02805                 if (activeVertices.Contains(prevVertex))
02806                 {
02807                     int currNext = newNexts[prevVertex];
02808                     while (!outPaths[prevVertex].Contains(currNext))
02809                     {
02810                         currNext = newNexts[currNext];
02811                     }
02812                     newNexts[prevVertex] = currNext;
02813                 }
02814             }
02815
02816             return polygons;
02817         }
02818
02819         private static IEnumerable<GraphicsPath> TriangulateMonotone(List<Point> vertices, List<(int,
```

```
    int)> edges, int[] directions, int targetSign)
02820         {
02821             int getDirection((int, int) edge)
02822             {
02823                 if (vertices[edge.Item2].Y != vertices[edge.Item1].Y)
02824                 {
02825                     return Math.Sign(vertices[edge.Item2].Y - vertices[edge.Item1].Y);
02826                 }
02827                 else
02828                 {
02829                     for (int i = 0; i < edges.Count; i++)
02830                     {
02831                         if (edges[i].Item2 == edge.Item1)
02832                         {
02833                             return getDirection(edges[i]);
02834                         }
02835                     }
02836                 }
02837 
02838                 throw new InvalidOperationException("Unknown edge direction!");
02839             }
02840 
02841 
02842             for (int i = 0; i < edges.Count; i++)
02843             {
02844                 directions[edges[i].Item1] = getDirection(edges[i]);
02845             }
02846 
02847             int[] sortedVertices = (from el in edges select el.Item1).OrderBy(a => a,
    Comparer<int>.Create((a, b) =>
02848             {
02849                 if (vertices[a].Y != vertices[b].Y)
02850                 {
02851                     return Math.Sign(vertices[a].Y - vertices[b].Y);
02852                 }
02853                 else
02854                 {
02855                     return Math.Sign(vertices[a].X - vertices[b].X);
02856                 }
02857             })).ToArray();
02858 
02859             List<(int, int)> diagonals = new List<(int, int)>();
02860 
02861             Stack<int> stack = new Stack<int>();
02862 
02863             stack.Push(sortedVertices[0]);
02864             stack.Push(sortedVertices[1]);
02865 
02866             for (int i = 2; i < sortedVertices.Length - 1; i++)
02867             {
02868                 int onTop = stack.Peek();
02869 
02870                 if (directions[sortedVertices[i]] != directions[onTop])
02871                 {
02872                     while (stack.Count > 1)
02873                     {
02874                         int v = stack.Pop();
02875 
02876                         diagonals.Add((v, sortedVertices[i]));
02877                     }
02878 
02879                     stack.Pop();
02880 
02881                     stack.Push(sortedVertices[i - 1]);
02882                     stack.Push(sortedVertices[i]);
02883                 }
02884                 else
02885                 {
02886                     int lastPopped = stack.Pop();
02887 
02888                     bool shouldContinue = true;
02889 
02890                     while (shouldContinue && stack.Count > 0)
02891                     {
02892                         int currVert = stack.Peek();
02893 
02894                         double areaSign = Math.Sign((vertices[currVert].X - vertices[lastPopped].X) *
    (vertices[sortedVertices[i]].Y - vertices[lastPopped].Y) - (vertices[currVert].Y -
    vertices[lastPopped].Y) * (vertices[sortedVertices[i]].X - vertices[lastPopped].X));
02895 
02896                         double dirSign = Math.Sign(vertices[currVert].X - vertices[lastPopped].X);
02897 
02898                         if (areaSign * dirSign > 0)
02899                         {
02900                             lastPopped = stack.Pop();
02901 
02902                             diagonals.Add((currVert, sortedVertices[i]));
```

```
02903                           }
02904                           else
02905                           {
02906                               shouldContinue = false;
02907                           }
02908                       }
02909 
02910                       stack.Push(lastPopped);
02911                       stack.Push(sortedVertices[i]);
02912                   }
02913               }
02914 
02915               stack.Pop();
02916 
02917               while (stack.Count > 1)
02918               {
02919                   int v = stack.Pop();
02920 
02921                   diagonals.Add((v, sortedVertices[sortedVertices.Length - 1]));
02922               }
02923 
02924               List<int>[] connections = new List<int>[vertices.Count];
02925 
02926               for (int i = 0; i < edges.Count; i++)
02927               {
02928                   connections[edges[i].Item1] = new List<int>();
02929               }
02930 
02931               for (int i = 0; i < edges.Count; i++)
02932               {
02933                   connections[edges[i].Item1].Add(edges[i].Item2);
02934                   connections[edges[i].Item2].Add(edges[i].Item1);
02935               }
02936 
02937               for (int i = 0; i < diagonals.Count; i++)
02938               {
02939                   connections[diagonals[i].Item1].Add(diagonals[i].Item2);
02940                   connections[diagonals[i].Item1].Add(diagonals[i].Item2);
02941 
02942                   connections[diagonals[i].Item2].Add(diagonals[i].Item1);
02943                   connections[diagonals[i].Item2].Add(diagonals[i].Item1);
02944               }
02945 
02946               int totalTriangles = (edges.Count + diagonals.Count * 2) / 3;
02947 
02948               List<List<(int, int)» polygons = new List<List<(int, int)»();
02949 
02950               while (polygons.Count < totalTriangles)
02951               {
02952                   int p1 = -1;
02953                   int p2 = -1;
02954 
02955                   for (int i = 0; i < connections.Length; i++)
02956                   {
02957                       if (connections[i] != null && connections[i].Count > 0)
02958                       {
02959                           p1 = i;
02960                           p2 = connections[i][0];
02961 
02962                           connections[i].Remove(p2);
02963                           connections[p2].Remove(i);
02964                           break;
02965                       }
02966                   }
02967 
02968                   int p3 = -1;
02969 
02970                   for (int i = 0; i < connections[p1].Count; i++)
02971                   {
02972                       if (connections[connections[p1][i]].Contains(p2))
02973                       {
02974                           p3 = connections[p1][i];
02975                           connections[p1].Remove(p3);
02976                           connections[p2].Remove(p3);
02977                           connections[p3].Remove(p1);
02978                           connections[p3].Remove(p2);
02979                           break;
02980                       }
02981                   }
02982 
02983                   int sign = Math.Sign((vertices[p1].X - vertices[p2].X) * (vertices[p3].Y -
      vertices[p2].Y) - (vertices[p1].Y - vertices[p2].Y) * (vertices[p3].X - vertices[p2].X));
02984 
02985                   if (sign == targetSign)
02986                   {
02987                       polygons.Add(new List<(int, int)>() { (p1, p2), (p2, p3), (p3, p1) });
02988                   }
```

```
02989                 else
02990                 {
02991                     polygons.Add(new List<(int, int)>() { (p1, p3), (p3, p2), (p2, p1) });
02992                 }
02993             }
02994 
02995             foreach (List<(int, int)> polygon in polygons)
02996             {
02997                 GraphicsPath polygonPath = new GraphicsPath();
02998                 foreach ((int, int) edge in polygon)
02999                 {
03000                     if (polygonPath.Segments.Count == 0)
03001                     {
03002                         polygonPath.MoveTo(vertices[edge.Item1]);
03003                     }
03004 
03005                     polygonPath.LineTo(vertices[edge.Item2]);
03006                 }
03007 
03008                 yield return polygonPath;
03009             }
03010         }
03011 
03012 /// <summary>
03013 /// Transforms all of the <see cref="Point"/>s in the <see cref="GraphicsPath"/> with an arbitrary
      transformation function.
03014 /// </summary>
03015 /// <param name="transformationFunction">An arbitrary transformation function.</param>
03016 /// <returns>A new <see cref="GraphicsPath"/> in which all points have been replaced using the
      <paramref name="transformationFunction"/>.</returns>
03017         [Pure]
03018         public GraphicsPath Transform(Func<Point, Point> transformationFunction)
03019         {
03020             GraphicsPath tbr = new GraphicsPath();
03021 
03022             foreach (Segment seg in this.Segments)
03023             {
03024                 tbr.Segments.AddRange(seg.Transform(transformationFunction));
03025             }
03026 
03027             return tbr;
03028         }
03029 
03030 
03031         private Rectangle cachedBounds = Rectangle.NaN;
03032 
03033 /// <summary>
03034 /// Compute the rectangular bounds of the path.
03035 /// </summary>
03036 /// <returns>The smallest <see cref="Rectangle"/> that contains the path.</returns>
03037         public Rectangle GetBounds()
03038         {
03039             if (double.IsNaN(cachedBounds.Location.X) || double.IsNaN(cachedBounds.Location.Y) ||
      double.IsNaN(cachedBounds.Size.Width) || double.IsNaN(cachedBounds.Size.Height))
03040             {
03041                 Point min = new Point(double.MaxValue, double.MaxValue);
03042                 Point max = new Point(double.MinValue, double.MinValue);
03043 
03044                 Point currPoint = new Point();
03045                 Point figureStartPoint = new Point();
03046 
03047                 for (int i = 0; i < this.Segments.Count; i++)
03048                 {
03049                     switch (this.Segments[i].Type)
03050                     {
03051                         case SegmentType.Move:
03052                             currPoint = this.Segments[i].Point;
03053                             figureStartPoint = this.Segments[i].Point;
03054                             min = Point.Min(min, this.Segments[i].Point);
03055                             max = Point.Max(max, this.Segments[i].Point);
03056                             break;
03057                         case SegmentType.Line:
03058                             if (i > 0)
03059                             {
03060                                 currPoint = this.Segments[i].Point;
03061                             }
03062                             else
03063                             {
03064                                 currPoint = this.Segments[i].Point;
03065                                 figureStartPoint = this.Segments[i].Point;
03066                             }
03067                             min = Point.Min(min, this.Segments[i].Point);
03068                             max = Point.Max(max, this.Segments[i].Point);
03069                             break;
03070                         case SegmentType.Arc:
03071                             ArcSegment seg = (ArcSegment)this.Segments[i];
03072 
```

```
03073                               if (i == 0)
03074                               {
03075                                   figureStartPoint = new Point(seg.Points[0].X +
      Math.Cos(seg.StartAngle) * seg.Radius, seg.Points[0].Y + Math.Sin(seg.StartAngle) * seg.Radius);
03076                                   currPoint = figureStartPoint;
03077
03078                                   min = Point.Min(min, currPoint);
03079                                   max = Point.Max(max, currPoint);
03080                               }
03081
03082                               double theta1 = seg.StartAngle;
03083                               double theta2 = seg.EndAngle;
03084
03085                               if (theta1 > theta2)
03086                               {
03087                                   (theta2, theta1) = (theta1, theta2);
03088                               }
03089
03090                               while (theta1 < 0)
03091                               {
03092                                   theta1 += 2 * Math.PI;
03093                                   theta2 += 2 * Math.PI;
03094                               }
03095
03096                               theta1 /= 2 * Math.PI;
03097                               theta2 /= 2 * Math.PI;
03098
03099                               int minAngle = (int)Math.Min(0, Math.Floor(theta1)) * 4;
03100                               int maxAngle = (int)Math.Max(1, Math.Ceiling(theta2)) * 4;
03101
03102                               theta1 *= 4;
03103                               theta2 *= 4;
03104
03105                               bool right = false;
03106                               bool bottom = false;
03107                               bool left = false;
03108                               bool top = false;
03109
03110                               for (int j = minAngle; j < maxAngle; j++)
03111                               {
03112                                   if (theta1 <= j && theta2 >= j)
03113                                   {
03114                                       switch (j % 4)
03115                                       {
03116                                           case 0:
03117                                               right = true;
03118                                               break;
03119                                           case 1:
03120                                               bottom = true;
03121                                               break;
03122                                           case 2:
03123                                               left = true;
03124                                               break;
03125                                           case 3:
03126                                               top = true;
03127                                               break;
03128                                       }
03129                                   }
03130                               }
03131
03132                               if (right)
03133                               {
03134                                   Point p = new Point(seg.Points[0].X + seg.Radius, seg.Points[0].Y);
03135                                   min = Point.Min(min, p);
03136                                   max = Point.Max(max, p);
03137                               }
03138
03139                               if (bottom)
03140                               {
03141                                   Point p = new Point(seg.Points[0].X, seg.Points[0].Y + seg.Radius);
03142                                   min = Point.Min(min, p);
03143                                   max = Point.Max(max, p);
03144                               }
03145
03146                               if (left)
03147                               {
03148                                   Point p = new Point(seg.Points[0].X - seg.Radius, seg.Points[0].Y);
03149                                   min = Point.Min(min, p);
03150                                   max = Point.Max(max, p);
03151                               }
03152
03153                               if (top)
03154                               {
03155                                   Point p = new Point(seg.Points[0].X, seg.Points[0].Y - seg.Radius);
03156                                   min = Point.Min(min, p);
03157                                   max = Point.Max(max, p);
03158                               }
```

```
03159
03160                               min = Point.Min(min, this.Segments[i].Point);
03161                               max = Point.Max(max, this.Segments[i].Point);
03162
03163                               currPoint = this.Segments[i].Point;
03164                               break;
03165                           case SegmentType.Close:
03166                               currPoint = figureStartPoint;
03167                               break;
03168                           case SegmentType.CubicBezier:
03169                               if (i == 0)
03170                               {
03171                                   currPoint = this.Segments[i].Points[0];
03172                                   figureStartPoint = this.Segments[i].Points[0];
03173                               }
03174
03175                               min = Point.Min(min, this.Segments[i].Point);
03176                               max = Point.Max(max, this.Segments[i].Point);
03177
03178                               {
03179                                   double aX = -currPoint.X + 3 * this.Segments[i].Points[0].X - 3 *
      this.Segments[i].Points[1].X + this.Segments[i].Point.X;
03180                                   double bX = 2 * (currPoint.X - 2 * this.Segments[i].Points[0].X +
      this.Segments[i].Points[1].X);
03181                                   double cX = this.Segments[i].Points[0].X - currPoint.X;
03182
03183                                   double t1X;
03184                                   double t2X;
03185
03186                                   if (Math.Abs(aX) < 1e-5)
03187                                   {
03188                                       if (Math.Abs(bX) >= 1e-5)
03189                                       {
03190                                           t1X = -cX / bX;
03191                                           t2X = t1X;
03192                                       }
03193                                       else
03194                                       {
03195                                           t1X = double.NaN;
03196                                           t2X = double.NaN;
03197                                       }
03198                                   }
03199                                   else
03200                                   {
03201                                       double delta = bX * bX - 4 * aX * cX;
03202
03203                                       if (delta >= -1e-5)
03204                                       {
03205                                           delta = Math.Max(delta, 0);
03206
03207                                           delta = Math.Sqrt(delta);
03208                                           t1X = (-bX + delta) / (2 * aX);
03209                                           t2X = (-bX - delta) / (2 * aX);
03210                                       }
03211                                       else
03212                                       {
03213                                           t1X = double.NaN;
03214                                           t2X = double.NaN;
03215                                       }
03216                                   }
03217
03218                                   if (t1X >= 0 && t1X <= 1)
03219                                   {
03220                                       Point p1 =
      ((CubicBezierSegment)this.Segments[i]).GetBezierPointAt(currPoint, t1X);
03221                                       min = Point.Min(min, p1);
03222                                       max = Point.Max(max, p1);
03223                                   }
03224
03225                                   if (t2X >= 0 && t2X <= 1)
03226                                   {
03227                                       Point p2 =
      ((CubicBezierSegment)this.Segments[i]).GetBezierPointAt(currPoint, t2X);
03228                                       min = Point.Min(min, p2);
03229                                       max = Point.Max(max, p2);
03230                                   }
03231                               }
03232
03233                               {
03234                                   double aY = -currPoint.Y + 3 * this.Segments[i].Points[0].Y - 3 *
      this.Segments[i].Points[1].Y + this.Segments[i].Point.Y;
03235                                   double bY = 2 * (currPoint.Y - 2 * this.Segments[i].Points[0].Y +
      this.Segments[i].Points[1].Y);
03236                                   double cY = this.Segments[i].Points[0].Y - currPoint.Y;
03237
03238                                   double t1Y;
03239                                   double t2Y;
```

```
03240
03241                                   if (Math.Abs(aY) < 1e-5)
03242                                   {
03243                                       if (Math.Abs(bY) >= 1e-5)
03244                                       {
03245                                           t1Y = -cY / bY;
03246                                           t2Y = t1Y;
03247                                       }
03248                                       else
03249                                       {
03250                                           t1Y = double.NaN;
03251                                           t2Y = double.NaN;
03252                                       }
03253                                   }
03254                                   else
03255                                   {
03256                                       double delta = bY * bY - 4 * aY * cY;
03257
03258                                       if (delta >= -1e-5)
03259                                       {
03260                                           delta = Math.Max(delta, 0);
03261
03262                                           delta = Math.Sqrt(delta);
03263                                           t1Y = (-bY + delta) / (2 * aY);
03264                                           t2Y = (-bY - delta) / (2 * aY);
03265                                       }
03266                                       else
03267                                       {
03268                                           t1Y = double.NaN;
03269                                           t2Y = double.NaN;
03270                                       }
03271                                   }
03272
03273                                   if (t1Y >= 0 && t1Y <= 1)
03274                                   {
03275                                       Point p1 =
      ((CubicBezierSegment)this.Segments[i]).GetBezierPointAt(currPoint, t1Y);
03276                                       min = Point.Min(min, p1);
03277                                       max = Point.Max(max, p1);
03278                                   }
03279
03280                                   if (t2Y >= 0 && t2Y <= 1)
03281                                   {
03282                                       Point p2 =
      ((CubicBezierSegment)this.Segments[i]).GetBezierPointAt(currPoint, t2Y);
03283                                       min = Point.Min(min, p2);
03284                                       max = Point.Max(max, p2);
03285                                   }
03286                               }
03287
03288                               currPoint = this.Segments[i].Point;
03289                               break;
03290                           }
03291                       }
03292
03293                       cachedBounds = new Rectangle(min, max);
03294                   }
03295
03296                   return cachedBounds;
03297               }
03298
03299               internal GraphicsPath ConvertArcsToBeziers()
03300               {
03301                   GraphicsPath tbr = new GraphicsPath();
03302
03303                   foreach (Segment seg in this.Segments)
03304                   {
03305                       if (seg is ArcSegment arc)
03306                       {
03307                           tbr.Segments.AddRange(arc.ToBezierSegments());
03308                       }
03309                       else
03310                       {
03311                           tbr.Segments.Add(seg);
03312                       }
03313                   }
03314
03315                   return tbr;
03316               }
03317
03318               private static Point IntersectLines(Point p1, Point d1, Point p3, Point d3)
03319               {
03320                   Point p2 = p1 + d1;
03321                   Point p4 = p3 + d3;
03322                   double denom = d1.X * d3.Y - d1.Y * d3.X;
03323
03324                   double x = (-(p1.X * p2.Y - p1.Y * p2.X) * d3.X + d1.X * (p3.X * p4.Y - p3.Y * p4.X)) /
```

```
denom;
03325             double y = (-(p1.X * p2.Y - p1.Y * p2.X) * d3.Y + d1.Y * (p3.X * p4.Y - p3.Y * p4.X)) /
denom;
03326
03327             return new Point(x, y);
03328         }
03329
03330 /// <summary>
03331 /// Reverses the <see cref="GraphicsPath"/>.
03332 /// </summary>
03333 /// <returns>The reversed <see cref="GraphicsPath"/>.</returns>
03334         [Pure]
03335         public GraphicsPath Reverse()
03336         {
03337             GraphicsPath tbr = new GraphicsPath();
03338
03339             foreach (GraphicsPath figure in this.GetFigures())
03340             {
03341                 Point currPoint = new Point();
03342                 Point startPoint = new Point();
03343                 bool started = false;
03344
03345                 List<(int type, Point p1, Point p2, Point p3, Point p4)> invertedSegments = new
List<(int type, Point p1, Point p2, Point p3, Point p4)>();
03346
03347                 for (int i = 0; i < figure.Segments.Count; i++)
03348                 {
03349                     if (figure.Segments[i].Type == SegmentType.Move)
03350                     {
03351                         currPoint = figure.Segments[i].Point;
03352                         startPoint = currPoint;
03353                         started = true;
03354                     }
03355                     else if (figure.Segments[i].Type == SegmentType.Line)
03356                     {
03357                         invertedSegments.Add((0, figure.Segments[i].Point, currPoint, new Point(), new
Point()));
03358                         currPoint = figure.Segments[i].Point;
03359
03360                         if (!started)
03361                         {
03362                             started = true;
03363                             startPoint = currPoint;
03364                         }
03365                     }
03366                     else if (figure.Segments[i].Type == SegmentType.Arc && figure.Segments[i] is
ArcSegment arc)
03367                     {
03368                         invertedSegments.Add((1, figure.Segments[i].Points[0], new Point(arc.Radius,
0), new Point(arc.EndAngle, arc.StartAngle), new Point()));
03369                         currPoint = figure.Segments[i].GetPointAt(currPoint, 1);
03370
03371                         if (!started)
03372                         {
03373                             started = true;
03374                             startPoint = figure.Segments[i].GetPointAt(currPoint, 0);
03375                         }
03376                     }
03377                     else if (figure.Segments[i].Type == SegmentType.CubicBezier)
03378                     {
03379                         invertedSegments.Add((2, figure.Segments[i].Points[2],
figure.Segments[i].Points[1], figure.Segments[i].Points[0], currPoint));
03380                         currPoint = figure.Segments[i].GetPointAt(currPoint, 1);
03381
03382                         if (!started)
03383                         {
03384                             started = true;
03385                             startPoint = figure.Segments[i].GetPointAt(currPoint, 0);
03386                         }
03387                     }
03388                     else if (figure.Segments[i].Type == SegmentType.Close)
03389                     {
03390                         invertedSegments.Add((3, startPoint, currPoint, new Point(), new Point()));
03391                     }
03392                 }
03393
03394                 bool isClosed = false;
03395
03396                 for (int i = invertedSegments.Count - 1; i >= 0; i--)
03397                 {
03398                     if (invertedSegments[i].type == 3)
03399                     {
03400                         tbr.MoveTo(invertedSegments[i].p1);
03401                         tbr.LineTo(invertedSegments[i].p2);
03402                         isClosed = true;
03403                     }
03404                     else if (invertedSegments[i].type == 0)
```

```
03405                     {
03406                         if (i == invertedSegments.Count - 1)
03407                         {
03408                             tbr.MoveTo(invertedSegments[i].p1);
03409                         }
03410 
03411                         tbr.LineTo(invertedSegments[i].p2);
03412                     }
03413                     else if (invertedSegments[i].type == 1)
03414                     {
03415                         tbr.Arc(invertedSegments[i].p1, invertedSegments[i].p2.X,
      invertedSegments[i].p3.X, invertedSegments[i].p3.Y);
03416                     }
03417                     else if (invertedSegments[i].type == 2)
03418                     {
03419                         if (i == invertedSegments.Count - 1)
03420                         {
03421                             tbr.MoveTo(invertedSegments[i].p1);
03422                         }
03423 
03424                         tbr.CubicBezierTo(invertedSegments[i].p2, invertedSegments[i].p3,
      invertedSegments[i].p4);
03425                     }
03426                 }
03427 
03428                 if (isClosed)
03429                 {
03430                     tbr.Close();
03431                 }
03432             }
03433 
03434             return tbr;
03435         }
03436 
03437 /// <summary>
03438 /// Returns a <see cref="GraphicsPath"/> representing the stroke of the current <see
      cref="GraphicsPath"/>.
03439 /// </summary>
03440 /// <param name="lineWidth">The thickness of the stroke.</param>
03441 /// <param name="lineCap">The line cap used in the stroke.</param>
03442 /// <param name="lineJoin">The line join used in the stroke.</param>
03443 /// <returns>A <see cref="GraphicsPath"/> representing the stroke of the current <see
      cref="GraphicsPath"/>.</returns>
03444         [Pure]
03445         public GraphicsPath GetStroke(double lineWidth = 1, LineCaps lineCap = LineCaps.Butt,
      LineJoins lineJoin = LineJoins.Miter)
03446         {
03447             double flatness = lineWidth * 0.0001;
03448 
03449             GraphicsPath tbr = new GraphicsPath();
03450 
03451             foreach (GraphicsPath figure in this.GetFigures())
03452             {
03453                 GraphicsPath half1 = new GraphicsPath();
03454 
03455                 ProcessHalfStroke(figure, half1, lineWidth, lineJoin, flatness, true);
03456 
03457                 tbr.AddPath(half1);
03458 
03459                 GraphicsPath half2 = new GraphicsPath();
03460                 ProcessHalfStroke(figure, half2, lineWidth, lineJoin, flatness, false);
03461 
03462                 half2 = half2.Reverse();
03463 
03464                 if (figure.Segments[figure.Segments.Count - 1].Type != SegmentType.Close &&
      half1.Segments.Count > 0 && half2.Segments.Count > 0)
03465                 {
03466                     if (lineCap == LineCaps.Butt)
03467                     {
03468                         if (half2.Segments[0].Type == SegmentType.Move)
03469                         {
03470                             half2.Segments[0] = new LineSegment(half2.Segments[0].Point);
03471                         }
03472 
03473                         tbr.AddPath(half2);
03474                         tbr.Close();
03475                     }
03476                     else if (lineCap == LineCaps.Square)
03477                     {
03478                         if (half2.Segments[0].Type == SegmentType.Move)
03479                         {
03480                             half2.Segments[0] = new LineSegment(half2.Segments[0].Point);
03481                         }
03482 
03483                         Point cap1 = tbr.GetPointAtRelative(1);
03484                         Point cap2 = half2.GetPointAtRelative(0);
03485 
```

```
03486                       Point tangent = this.GetTangentAtRelative(1);
03487
03488                       tbr.LineTo(cap1 + tangent * lineWidth * 0.5);
03489                       tbr.LineTo(cap2 + tangent * lineWidth * 0.5);
03490
03491                       tbr.AddPath(half2);
03492
03493                       Point cap3 = tbr.GetPointAtRelative(0);
03494                       Point cap4 = half2.GetPointAtRelative(1);
03495
03496                       Point tangent2 = this.GetTangentAtRelative(0) * -1;
03497
03498                       tbr.LineTo(cap4 + tangent2 * lineWidth * 0.5);
03499                       tbr.LineTo(cap3 + tangent2 * lineWidth * 0.5);
03500
03501                       tbr.Close();
03502                   }
03503                   else if (lineCap == LineCaps.Round)
03504                   {
03505                       if (half2.Segments[0].Type == SegmentType.Move)
03506                       {
03507                           half2.Segments[0] = new LineSegment(half2.Segments[0].Point);
03508                       }
03509
03510                       Point cap1 = tbr.GetPointAtRelative(1);
03511                       Point cap2 = half2.GetPointAtRelative(0);
03512
03513                       Point center = this.GetPointAtRelative(1);
03514
03515                       double ang1 = (Math.Atan2(cap1.Y - center.Y, cap1.X - center.X) + 2 * Math.PI)
      % (2 * Math.PI);
03516                       double ang2 = (Math.Atan2(cap2.Y - center.Y, cap2.X - center.X) + 2 * Math.PI)
      % (2 * Math.PI);
03517
03518                       tbr.Arc(center, lineWidth * 0.5, ang1, ang2);
03519
03520
03521                       tbr.AddPath(half2);
03522
03523                       Point cap3 = tbr.GetPointAtRelative(0);
03524                       Point cap4 = half2.GetPointAtRelative(1);
03525
03526                       Point center2 = this.GetPointAtRelative(0);
03527
03528                       double ang3 = (Math.Atan2(cap4.Y - center2.Y, cap4.X - center2.X) + 2 *
      Math.PI) % (2 * Math.PI);
03529                       double ang4 = (Math.Atan2(cap3.Y - center2.Y, cap3.X - center2.X) + 2 *
      Math.PI) % (2 * Math.PI);
03530
03531                       tbr.Arc(center2, lineWidth * 0.5, ang3, ang4);
03532
03533                       tbr.Close();
03534                   }
03535               }
03536               else
03537               {
03538                   tbr.AddPath(half2);
03539               }
03540           }
03541
03542           return RemoveDuplicates(tbr);
03543       }
03544
03545       // Adapted from https://www.particleincell.com/2013/cubic-line-intersection/
03546       private static double[] CubicRoots(double a, double b, double c, double d)
03547       {
03548           double A = b / a;
03549           double B = c / a;
03550           double C = d / a;
03551
03552           double Q = (3 * B - A * A) / 9;
03553           double R = (9 * A * B - 27 * C - 2 * A * A * A) / 54;
03554           double D = Q * Q * Q + R * R; // polynomial discriminant
03555
03556           if (Math.Abs(D) < Tolerance)
03557           {
03558               D = 0;
03559           }
03560
03561           double[] t = new double[3];
03562
03563           if (D >= 0)                                 // complex or duplicate roots
03564           {
03565               var S = Math.Sign(R + Math.Sqrt(D)) * Math.Pow(Math.Abs(R + Math.Sqrt(D)), (1.0 / 3));
03566               var T = Math.Sign(R - Math.Sqrt(D)) * Math.Pow(Math.Abs(R - Math.Sqrt(D)), (1.0 / 3));
03567
03568               t[0] = -A / 3 + (S + T);                    // real root
```

```
03569                 t[1] = -A / 3 - (S + T) / 2;                    // real part of complex root
03570                 t[2] = -A / 3 - (S + T) / 2;                    // real part of complex root
03571                 double Im = Math.Abs(Math.Sqrt(3) * (S - T) / 2);    // complex part of root pair
03572 
03573                 /*discard complex roots*/
03574                 if (Im > Tolerance)
03575                 {
03576                     t[1] = -1;
03577                     t[2] = -1;
03578                 }
03579 
03580             }
03581             else                                          // distinct real roots
03582             {
03583                 var th = Math.Acos(R / Math.Sqrt(-Math.Pow(Q, 3)));
03584 
03585                 t[0] = 2 * Math.Sqrt(-Q) * Math.Cos(th / 3) - A / 3;
03586                 t[1] = 2 * Math.Sqrt(-Q) * Math.Cos((th + 2 * Math.PI) / 3) - A / 3;
03587                 t[2] = 2 * Math.Sqrt(-Q) * Math.Cos((th + 4 * Math.PI) / 3) - A / 3;
03588             }
03589 
03590             return new HashSet<double>(t.Select(x => Math.Abs(x) <= Tolerance ?  0 :  Math.Abs(x - 1)
      <= Tolerance ?  1 :  x).Where(x => x >= 0 && x <= 1)).ToArray();
03591         }
03592 
03593 /// <summary>
03594 /// Determines whether the specified <paramref name="point"/> falls within the current <see
      cref="GraphicsPath"/>.
03595 /// </summary>
03596 /// <param name="point">The <see cref="Point"/> being analysed.</param>
03597 /// <param name="fillRule">The rule to use to determine whether a point is inside or outside of a the
      <see cref="GraphicsPath"/>.</param>
03598 /// <returns><see langword="true"/> if the <paramref name="point"/> is inside the <see
      cref="GraphicsPath"/>, or <see langword="false"/> if it is outside.  If the point lies exactly on the
      <see cref="GraphicsPath"/> (or very close to it), the result may be either <see langword="true"/> or
      <see langword="false"/>, depending on numerical approximations.</returns>
03599         public bool ContainsPoint(Point point, FillRule fillRule)
03600         {
03601             return ContainsPointInternal(point, fillRule == FillRule.EvenOdd) > 0;
03602         }
03603 
03604         private GraphicsPath MonotoniseOnY()
03605         {
03606             GraphicsPath tbr = new GraphicsPath();
03607             Point startPoint = new Point();
03608             Point currPoint = new Point();
03609 
03610 
03611             foreach (Segment seg in this.Segments)
03612             {
03613                 switch (seg.Type)
03614                 {
03615                     case SegmentType.Move:
03616                         tbr.Segments.Add(seg);
03617                         currPoint = seg.Point;
03618                         startPoint = seg.Point;
03619                         break;
03620 
03621                     case SegmentType.Line:
03622                         tbr.Segments.Add(seg);
03623                         currPoint = seg.Point;
03624                         break;
03625 
03626                     case SegmentType.Close:
03627                         tbr.Segments.Add(seg);
03628                         currPoint = startPoint;
03629                         break;
03630 
03631                     case SegmentType.Arc:
03632                         foreach (Segment seg2 in ((ArcSegment)seg).ToBezierSegments())
03633                         {
03634                             if (seg2 is CubicBezierSegment cub)
03635                             {
03636                                 tbr.Segments.AddRange(cub.MonotoniseOnY(currPoint));
03637                                 currPoint = tbr.Segments[tbr.Segments.Count - 1].Point;
03638                             }
03639                             else
03640                             {
03641                                 tbr.Segments.Add(seg2);
03642                                 currPoint = tbr.Segments[tbr.Segments.Count - 1].Point;
03643                             }
03644                         }
03645                         break;
03646                     case SegmentType.CubicBezier:
03647                         tbr.Segments.AddRange(((CubicBezierSegment)seg).MonotoniseOnY(currPoint));
03648                         currPoint = tbr.Segments[tbr.Segments.Count - 1].Point;
03649                         break;
```

```
03650                 }
03651             }
03652
03653             return tbr;
03654         }
03655
03656         private GraphicsPath Monotonise(double flatness)
03657         {
03658             GraphicsPath tbr = new GraphicsPath();
03659             Point startPoint = new Point();
03660             Point currPoint = new Point();
03661
03662             foreach (Segment seg in this.Segments)
03663             {
03664                 switch (seg.Type)
03665                 {
03666                     case SegmentType.Move:
03667                         tbr.Segments.Add(seg);
03668                         currPoint = seg.Point;
03669                         startPoint = seg.Point;
03670                         break;
03671
03672                     case SegmentType.Line:
03673                         tbr.Segments.Add(seg);
03674                         currPoint = seg.Point;
03675                         break;
03676
03677                     case SegmentType.Close:
03678                         tbr.Segments.Add(seg);
03679                         currPoint = startPoint;
03680                         break;
03681
03682                     case SegmentType.Arc:
03683                         foreach (Segment seg2 in ((ArcSegment)seg).ToBezierSegments())
03684                         {
03685                             if (seg2 is CubicBezierSegment cub)
03686                             {
03687                                 tbr.Segments.AddRange(cub.BreakAtInflectionPoints(currPoint,
      flatness));
03688                                 currPoint = tbr.Segments[tbr.Segments.Count - 1].Point;
03689                             }
03690                             else
03691                             {
03692                                 tbr.Segments.Add(seg2);
03693                                 currPoint = tbr.Segments[tbr.Segments.Count - 1].Point;
03694                             }
03695                         }
03696                         break;
03697                     case SegmentType.CubicBezier:
03698
      tbr.Segments.AddRange(((CubicBezierSegment)seg).BreakAtInflectionPoints(currPoint, flatness));
03699                         currPoint = tbr.Segments[tbr.Segments.Count - 1].Point;
03700                         break;
03701                 }
03702             }
03703
03704             return tbr;
03705         }
03706
03707         private int ContainsPointInternal(Point point, bool evenOdd)
03708         {
03709             int countsAbs = 0;
03710             int counts = 0;
03711
03712             foreach (GraphicsPath figure in this.MonotoniseOnY().GetFigures())
03713             {
03714                 if (figure.Segments[figure.Segments.Count - 1].Type == SegmentType.Close)
03715                 {
03716                     if (figure.Segments[0].Type != SegmentType.Move)
03717                     {
03718                         throw new InvalidOperationException("Assertion failed:  the figure does not
      start with a move segment!");
03719                     }
03720
03721                     (int, int)[] pointDirections = new (int, int)[figure.Segments.Count - 1];
03722
03723                     Point startPoint = new Point();
03724                     Point currPoint = new Point();
03725                     int index = 0;
03726
03727                     for (int i = 0; i < figure.Segments.Count; i++)
03728                     {
03729                         if (figure.Segments[i].Type == SegmentType.Move)
03730                         {
03731                             currPoint = figure.Segments[i].Point;
03732                             startPoint = currPoint;
03733                         }
```

```
03734                           else if (figure.Segments[i].Type == SegmentType.Line)
03735                           {
03736                               int direction = Math.Sign(figure.Segments[i].Point.Y - currPoint.Y);
03737 
03738                               pointDirections[index] = (pointDirections[index].Item1, direction);
03739                               pointDirections[index + 1] = (direction, pointDirections[index +
      1].Item2);
03740 
03741                               currPoint = figure.Segments[i].Point;
03742                               index++;
03743                           }
03744                           else if (figure.Segments[i].Type == SegmentType.Arc)
03745                           {
03746                               throw new InvalidOperationException("Path not properly monotonised!");
03747                           }
03748                           else if (figure.Segments[i].Type == SegmentType.CubicBezier)
03749                           {
03750                               int direction = Math.Sign(figure.Segments[i].Point.Y - currPoint.Y);
03751 
03752                               pointDirections[index] = (pointDirections[index].Item1, direction);
03753                               pointDirections[index + 1] = (direction, pointDirections[index +
      1].Item2);
03754 
03755                               currPoint = figure.Segments[i].Point;
03756                               index++;
03757                           }
03758                           else if (figure.Segments[i].Type == SegmentType.Close)
03759                           {
03760                               int direction = Math.Sign(startPoint.Y - currPoint.Y);
03761 
03762                               pointDirections[index] = (pointDirections[index].Item1, direction);
03763                               pointDirections[0] = (direction, pointDirections[0].Item2);
03764                           }
03765                       }
03766 
03767                       int[] directions = new int[pointDirections.Length * 2];
03768 
03769                       for (int i = 0; i < pointDirections.Length; i++)
03770                       {
03771                           directions[2 * i] = pointDirections[i].Item1;
03772                           directions[2 * i + 1] = pointDirections[i].Item2;
03773                       }
03774 
03775                       for (int i = 0; i < directions.Length; i++)
03776                       {
03777                           if (directions[i] == 0)
03778                           {
03779                               int nextDir = 0;
03780 
03781                               int j = 0;
03782                               while (directions[(i + j) % directions.Length] == 0)
03783                               {
03784                                   j++;
03785                               }
03786                               nextDir = directions[(i + j) % directions.Length];
03787 
03788                               int prevDir = 0;
03789 
03790                               j = 0;
03791                               while (directions[(i - j) >= 0 ?  (i - j) :  (i - j + directions.Length)]
      == 0)
03792                               {
03793                                   j++;
03794                               }
03795                               prevDir = directions[(i - j) >= 0 ?  (i - j) :  (i - j +
      directions.Length)];
03796 
03797                               if (prevDir == nextDir)
03798                               {
03799                                   directions[i] = prevDir;
03800                               }
03801                           }
03802                       }
03803 
03804                       for (int i = 0; i < pointDirections.Length; i++)
03805                       {
03806                           pointDirections[i] = (directions[2 * i], directions[2 * i + 1]);
03807                       }
03808 
03809                       startPoint = new Point();
03810                       currPoint = new Point();
03811                       index = 0;
03812 
03813                       for (int i = 0; i < figure.Segments.Count; i++)
03814                       {
03815                           if (figure.Segments[i].Type == SegmentType.Move)
03816                           {
```

```
03817                               currPoint = figure.Segments[i].Point;
03818                               startPoint = currPoint;
03819                           }
03820                           else if (figure.Segments[i].Type == SegmentType.Line)
03821                           {
03822                               if (currPoint.Y != figure.Segments[i].Point.Y)
03823                               {
03824                                   double t = (point.Y - currPoint.Y) / (figure.Segments[i].Point.Y -
      currPoint.Y);
03825
03826                                   if (t >= 0 && t <= 1)
03827                                   {
03828                                       double x = currPoint.X + (figure.Segments[i].Point.X -
      currPoint.X) * t;
03829
03830                                       if (x >= point.X)
03831                                       {
03832                                           if (t == 0)
03833                                           {
03834                                               if (!((pointDirections[index].Item1 == -1 &&
      pointDirections[index].Item2 == -1) || (pointDirections[index].Item1 == 0 &&
      pointDirections[index].Item2 == 0)))
03835                                               {
03836                                                   countsAbs++;
03837                                                   counts += pointDirections[index].Item2;
03838                                               }
03839                                           }
03840                                           else if (t == 1)
03841                                           {
03842                                               if (!((pointDirections[index + 1].Item1 == 1 &&
      pointDirections[index + 1].Item2 == 1) || (pointDirections[index + 1].Item1 == 0 &&
      pointDirections[index + 1].Item2 == 0)))
03843                                               {
03844                                                   countsAbs++;
03845                                                   counts += pointDirections[index + 1].Item1;
03846                                               }
03847                                           }
03848                                           else
03849                                           {
03850                                               countsAbs++;
03851                                               counts += Math.Sign(figure.Segments[i].Point.Y -
      currPoint.Y);
03852                                           }
03853                                       }
03854                                   }
03855                               }
03856                               /* else if (currPoint.Y == point.Y)
03857 {
03858 if (currPoint.X >= point.X)
03859 {
03860 if (!((pointDirections[index].Item1 == -1 && pointDirections[index].Item2 == -1) ||
      (pointDirections[index].Item1 == 0 && pointDirections[index].Item2 == 0)))
03861 {
03862 countsAbs++;
03863 counts += pointDirections[index].Item2;
03864 }
03865 }
03866
03867 if (figure.Segments[i].Point.X >= point.X)
03868 {
03869 if (!((pointDirections[index + 1].Item1 == 1 && pointDirections[index + 1].Item2 == 1) ||
      (pointDirections[index + 1].Item1 == 0 && pointDirections[index + 1].Item2 == 0)))
03870 {
03871 countsAbs++;
03872 counts += pointDirections[index + 1].Item1;
03873 }
03874 }
03875 }*/
03876
03877                               currPoint = figure.Segments[i].Point;
03878                               index++;
03879                           }
03880                           else if (figure.Segments[i].Type == SegmentType.CubicBezier)
03881                           {
03882                               double a = -currPoint.Y + 3 * figure.Segments[i].Points[0].Y - 3 *
      figure.Segments[i].Points[1].Y + figure.Segments[i].Points[2].Y;
03883                               double b = 3 * currPoint.Y - 6 * figure.Segments[i].Points[0].Y + 3 *
      figure.Segments[i].Points[1].Y;
03884                               double c = -3 * currPoint.Y + 3 * figure.Segments[i].Points[0].Y;
03885
03886
03887                               double d = currPoint.Y - point.Y;
03888
03889                               double[] roots = CubicRoots(a, b, c, d);
03890
03891                               if (roots.Length == 1)
03892                               {
```

```
03893                                 double t = roots[0];
03894                                 double x = (1 - t) * (1 - t) * (1 - t) * currPoint.X +
      figure.Segments[i].Points[0].X * t * (1 - t) * (1 - t) * 3 + figure.Segments[i].Points[1].X * t * t *
      (1 - t) * 3 + figure.Segments[i].Points[2].X * t * t * t;
03895
03896                                 if (x >= point.X)
03897                                 {
03898                                     if (t == 0)
03899                                     {
03900                                         if (!((pointDirections[index].Item1 == -1 &&
      pointDirections[index].Item2 == -1) || (pointDirections[index].Item1 == 0 &&
      pointDirections[index].Item2 == 0)))
03901                                         {
03902                                             countsAbs++;
03903                                             counts += pointDirections[index].Item2;
03904                                         }
03905                                     }
03906                                     else if (t == 1)
03907                                     {
03908                                         if (!((pointDirections[index + 1].Item1 == 1 &&
      pointDirections[index + 1].Item2 == 1) || (pointDirections[index + 1].Item1 == 0 &&
      pointDirections[index + 1].Item2 == 0)))
03909                                         {
03910                                             countsAbs++;
03911                                             counts += pointDirections[index + 1].Item1;
03912                                         }
03913                                     }
03914                                     else
03915                                     {
03916                                         countsAbs++;
03917                                         counts += Math.Sign(figure.Segments[i].Point.Y - currPoint.Y);
03918                                     }
03919                                 }
03920                             }
03921                             else if (roots.Length != 0)
03922                             {
03923                                 throw new InvalidOperationException("Path not properly monotonised!");
03924                             }
03925
03926                             currPoint = figure.Segments[i].Point;
03927                             index++;
03928                         }
03929                         else if (figure.Segments[i].Type == SegmentType.Close)
03930                         {
03931                             if (currPoint.Y != startPoint.Y)
03932                             {
03933                                 double t = (point.Y - currPoint.Y) / (startPoint.Y - currPoint.Y);
03934
03935                                 if (t >= 0 && t <= 1)
03936                                 {
03937                                     double x = currPoint.X + (startPoint.X - currPoint.X) * t;
03938
03939                                     if (x >= point.X)
03940                                     {
03941                                         if (t == 0)
03942                                         {
03943                                             if (!((pointDirections[index].Item1 == -1 &&
      pointDirections[index].Item2 == -1) || (pointDirections[index].Item1 == 0 &&
      pointDirections[index].Item2 == 0)))
03944                                             {
03945                                                 countsAbs++;
03946                                                 counts += pointDirections[index].Item2;
03947                                             }
03948                                         }
03949                                         else if (t == 1)
03950                                         {
03951                                             if (!((pointDirections[0].Item1 == 1 &&
      pointDirections[0].Item2 == 1) || (pointDirections[0].Item1 == 0 && pointDirections[0].Item2 == 0)))
03952                                             {
03953                                                 countsAbs++;
03954                                                 counts += pointDirections[0].Item1;
03955                                             }
03956                                         }
03957                                         else
03958                                         {
03959                                             countsAbs++;
03960                                             counts += Math.Sign(startPoint.Y - currPoint.Y);
03961                                         }
03962                                     }
03963                                 }
03964                             }
03965                             /* else if (currPoint.Y == point.Y)
03966 {
03967 if (currPoint.X >= point.X)
03968 {
03969 if (!((pointDirections[index].Item1 == -1 && pointDirections[index].Item2 == -1) ||
      (pointDirections[index].Item1 == 0 && pointDirections[index].Item2 == 0)))
```

```
03970 {
03971 countsAbs++;
03972 counts += pointDirections[index].Item2;
03973 }
03974 }
03975
03976 if (startPoint.X >= point.X)
03977 {
03978 if (!((pointDirections[0].Item1 == 1 && pointDirections[0].Item2 == 1) || (pointDirections[0].Item1 ==
      0 && pointDirections[0].Item2 == 0)))
03979 {
03980 countsAbs++;
03981 counts += pointDirections[0].Item1;
03982 }
03983 }
03984 }*/
03985                         }
03986                     }
03987                 }
03988             }
03989
03990             if (evenOdd)
03991             {
03992                 return countsAbs % 2;
03993             }
03994             else
03995             {
03996                 return counts != 0 ?  1 :  0;
03997             }
03998         }
03999
04000         private static void ProcessHalfStroke(GraphicsPath figure, GraphicsPath tbr, double lineWidth,
      LineJoins lineJoin, double flatness, bool positive)
04001         {
04002             Point currPoint = new Point();
04003             Point currTangent = new Point(double.NaN, double.NaN);
04004             double sign = positive ?  1 :  -1;
04005
04006             bool started = false;
04007             Point startPoint = new Point();
04008
04009             bool isClosed = false;
04010
04011             figure = RemoveDuplicates(figure.Monotonise(flatness));
04012
04013             foreach (Segment seg in figure.Segments)
04014             {
04015                 if (seg.Type == SegmentType.Move)
04016                 {
04017                     currPoint = seg.Point;
04018                     currTangent = new Point(double.NaN, double.NaN);
04019                     startPoint = currPoint;
04020                     started = true;
04021                 }
04022                 else if (seg.Type == SegmentType.Line)
04023                 {
04024                     Point segStartPoint = currPoint;
04025                     Point segTangent = seg.GetTangentAt(segStartPoint, 0);
04026                     Point segEndPoint = seg.Point;
04027
04028                     if (!double.IsNaN(currTangent.X) && !double.IsNaN(currTangent.Y) && (currTangent.X
      != segTangent.X && currTangent.Y != segTangent.Y) && !double.IsNaN(segTangent.X) &&
      !double.IsNaN(segTangent.Y))
04029                     {
04030                         Point p = currPoint;
04031                         Point pALeft = currPoint - new Point(-currTangent.Y, currTangent.X) *
      lineWidth * 0.5;
04032                         Point pARight = currPoint + new Point(-currTangent.Y, currTangent.X) *
      lineWidth * 0.5;
04033
04034                         Point pBLeft = currPoint - new Point(-segTangent.Y, segTangent.X) * lineWidth
      * 0.5;
04035                         Point pBRight = currPoint + new Point(-segTangent.Y, segTangent.X) * lineWidth
      * 0.5;
04036
04037                         Point p3_1 = IntersectLines(pALeft, currTangent, pBLeft, segTangent);
04038                         Point p3_2 = IntersectLines(pARight, currTangent, pBRight, segTangent);
04039
04040                         double crossProduct = currTangent.X * segTangent.Y - currTangent.Y *
      segTangent.X;
04041
04042                         if (crossProduct > 0 && !positive)
04043                         {
04044                             if (lineJoin == LineJoins.Round)
04045                             {
04046                                 tbr.Arc(p, lineWidth * 0.5, Math.Atan2(pALeft.Y - p.Y, pALeft.X -
      p.X), Math.Atan2(pBLeft.Y - p.Y, pBLeft.X - p.X));
```

```
04047                           }
04048                           else if (lineJoin == LineJoins.Miter)
04049                           {
04050                               tbr.LineTo(p3_1);
04051                           }
04052                       }
04053                       else if (crossProduct < 0 && positive)
04054                       {
04055                           if (lineJoin == LineJoins.Round)
04056                           {
04057                               tbr.Arc(p, lineWidth * 0.5, Math.Atan2(pARight.Y - p.Y, pARight.X -
      p.X), Math.Atan2(pBRight.Y - p.Y, pBRight.X - p.X));
04058                           }
04059                           else if (lineJoin == LineJoins.Miter)
04060                           {
04061                               tbr.LineTo(p3_2);
04062                           }
04063                       }
04064                   }
04065
04066                   if (!double.IsNaN(segTangent.X) && !double.IsNaN(segTangent.Y))
04067                   {
04068                       tbr.LineTo(currPoint + sign * lineWidth * 0.5 * new Point(-segTangent.Y,
      segTangent.X));
04069                       tbr.LineTo(seg.Point + sign * lineWidth * 0.5 * new Point(-segTangent.Y,
      segTangent.X));
04070                   }
04071
04072                   currPoint = seg.Point;
04073                   currTangent = segTangent;
04074
04075                   if (!started)
04076                   {
04077                       startPoint = currPoint;
04078                       started = true;
04079                   }
04080               }
04081               else if (seg.Type == SegmentType.Arc && seg is ArcSegment arc)
04082               {
04083                   Point segStartPoint = currPoint;
04084                   Point segTangent = seg.GetTangentAt(segStartPoint, 0);
04085                   Point segEndPoint = seg.Point;
04086
04087                   if (!double.IsNaN(currTangent.X) && !double.IsNaN(currTangent.Y) && (currTangent.X
      != segTangent.X && currTangent.Y != segTangent.Y) && !double.IsNaN(segTangent.X) &&
      !double.IsNaN(segTangent.Y))
04088                   {
04089                       Point p = currPoint;
04090                       Point pALeft = currPoint - new Point(-currTangent.Y, currTangent.X) *
      lineWidth * 0.5;
04091                       Point pARight = currPoint + new Point(-currTangent.Y, currTangent.X) *
      lineWidth * 0.5;
04092
04093                       Point pBLeft = currPoint - new Point(-segTangent.Y, segTangent.X) * lineWidth
      * 0.5;
04094                       Point pBRight = currPoint + new Point(-segTangent.Y, segTangent.X) * lineWidth
      * 0.5;
04095
04096                       Point p3_1 = IntersectLines(pALeft, currTangent, pBLeft, segTangent);
04097                       Point p3_2 = IntersectLines(pARight, currTangent, pBRight, segTangent);
04098
04099                       double crossProduct = currTangent.X * segTangent.Y - currTangent.Y *
      segTangent.X;
04100                       if (crossProduct > 0 && !positive)
04101                       {
04102                           if (lineJoin == LineJoins.Round)
04103                           {
04104                               tbr.Arc(p, lineWidth * 0.5, Math.Atan2(pALeft.Y - p.Y, pALeft.X -
      p.X), Math.Atan2(pBLeft.Y - p.Y, pBLeft.X - p.X));
04105                           }
04106                           else if (lineJoin == LineJoins.Miter)
04107                           {
04108                               tbr.LineTo(p3_1);
04109                           }
04110                       }
04111                       else if (crossProduct < 0 && positive)
04112                       {
04113                           if (lineJoin == LineJoins.Round)
04114                           {
04115                               tbr.Arc(p, lineWidth * 0.5, Math.Atan2(pARight.Y - p.Y, pARight.X -
      p.X), Math.Atan2(pBRight.Y - p.Y, pBRight.X - p.X));
04116                           }
04117                           else if (lineJoin == LineJoins.Miter)
04118                           {
04119                               tbr.LineTo(p3_2);
04120                           }
04121                       }
```

```
04122                     }
04123
04124                     tbr.LineTo(currPoint + sign * lineWidth * 0.5 * new Point(-segTangent.Y,
      segTangent.X));
04125                     tbr.Arc(arc.Points[0], arc.Radius - sign * lineWidth * 0.5 *
      Math.Sign(arc.EndAngle - arc.StartAngle), arc.StartAngle, arc.EndAngle);
04126
04127                     currPoint = seg.Point;
04128                     currTangent = seg.GetTangentAt(segStartPoint, 1);
04129
04130                     if (!started)
04131                     {
04132                         startPoint = currPoint;
04133                         started = true;
04134                     }
04135                 }
04136                 else if (seg.Type == SegmentType.CubicBezier)
04137                 {
04138                     Point segStartPoint = currPoint;
04139                     Point segTangent = seg.GetTangentAt(segStartPoint, 0);
04140                     Point segEndPoint = seg.Point;
04141
04142                     if (!double.IsNaN(currTangent.X) && !double.IsNaN(currTangent.Y) && (currTangent.X
      != segTangent.X && currTangent.Y != segTangent.Y) && !double.IsNaN(segTangent.X) &&
      !double.IsNaN(segTangent.Y))
04143                     {
04144                         Point p = currPoint;
04145                         Point pALeft = currPoint - new Point(-currTangent.Y, currTangent.X) *
      lineWidth * 0.5;
04146                         Point pARight = currPoint + new Point(-currTangent.Y, currTangent.X) *
      lineWidth * 0.5;
04147
04148                         Point pBLeft = currPoint - new Point(-segTangent.Y, segTangent.X) * lineWidth
      * 0.5;
04149                         Point pBRight = currPoint + new Point(-segTangent.Y, segTangent.X) * lineWidth
      * 0.5;
04150
04151                         Point p3_1 = IntersectLines(pALeft, currTangent, pBLeft, segTangent);
04152                         Point p3_2 = IntersectLines(pARight, currTangent, pBRight, segTangent);
04153
04154                         double crossProduct = currTangent.X * segTangent.Y - currTangent.Y *
      segTangent.X;
04155
04156                         if (crossProduct > 1e-7 && !positive)
04157                         {
04158                             if (lineJoin == LineJoins.Round)
04159                             {
04160                                 tbr.Arc(p, lineWidth * 0.5, Math.Atan2(pALeft.Y - p.Y, pALeft.X -
      p.X), Math.Atan2(pBLeft.Y - p.Y, pBLeft.X - p.X));
04161                             }
04162                             else if (lineJoin == LineJoins.Miter)
04163                             {
04164                                 tbr.LineTo(p3_1);
04165                             }
04166                         }
04167                         else if (crossProduct < -1e-7 && positive)
04168                         {
04169                             if (lineJoin == LineJoins.Round)
04170                             {
04171                                 tbr.Arc(p, lineWidth * 0.5, Math.Atan2(pARight.Y - p.Y, pARight.X -
      p.X), Math.Atan2(pBRight.Y - p.Y, pBRight.X - p.X));
04172                             }
04173                             else if (lineJoin == LineJoins.Miter)
04174                             {
04175                                 tbr.LineTo(p3_2);
04176                             }
04177                         }
04178                     }
04179
04180                     if (!double.IsNaN(segTangent.X) && !double.IsNaN(segTangent.Y))
04181                     {
04182                         tbr.LineTo(currPoint + sign * lineWidth * 0.5 * new Point(-segTangent.Y,
      segTangent.X));
04183                     }
04184
04185                     List<(Point point, Point tangent)> newPoints =
      seg.FlattenForOffsetAndGetTangents(currPoint, sign * lineWidth, flatness).ToList();
04186
04187                     for (int i = 0; i < newPoints.Count; i++)
04188                     {
04189                         Point pt = newPoints[i].point + sign * lineWidth * 0.5 * new
      Point(-newPoints[i].tangent.Y, newPoints[i].tangent.X);
04190
04191                         if (!double.IsNaN(pt.X) && !double.IsNaN(pt.Y))
04192                         {
04193                             tbr.LineTo(pt);
04194                         }
```

```
04195                     }
04196
04197                     if ((currPoint.X != newPoints[newPoints.Count - 1].point.X || currPoint.Y !=
      newPoints[newPoints.Count - 1].point.Y) || double.IsNaN(currTangent.X) || double.IsNaN(currTangent.Y))
04198                     {
04199                         currPoint = newPoints[newPoints.Count - 1].point;
04200                         currTangent = newPoints[newPoints.Count - 1].tangent;
04201                     }
04202
04203                     if (!started)
04204                     {
04205                         startPoint = currPoint;
04206                         started = true;
04207                     }
04208                 }
04209                 else if (seg.Type == SegmentType.Close)
04210                 {
04211                     Point segStartPoint = currPoint;
04212                     Point segTangent = new Point(startPoint.X - currPoint.X, startPoint.Y -
      currPoint.Y).Normalize();
04213                     Point segEndPoint = startPoint;
04214
04215                     if (!currPoint.IsEqual(startPoint, Tolerance))
04216                     {
04217                         if (!double.IsNaN(currTangent.X) && !double.IsNaN(currTangent.Y) &&
      (currTangent.X != segTangent.X && currTangent.Y != segTangent.Y))
04218                         {
04219                             Point p = currPoint;
04220                             Point pALeft = currPoint - new Point(-currTangent.Y, currTangent.X) *
      lineWidth * 0.5;
04221                             Point pARight = currPoint + new Point(-currTangent.Y, currTangent.X) *
      lineWidth * 0.5;
04222
04223                             Point pBLeft = currPoint - new Point(-segTangent.Y, segTangent.X) *
      lineWidth * 0.5;
04224                             Point pBRight = currPoint + new Point(-segTangent.Y, segTangent.X) *
      lineWidth * 0.5;
04225
04226                             Point p3_1 = IntersectLines(pALeft, currTangent, pBLeft, segTangent);
04227                             Point p3_2 = IntersectLines(pARight, currTangent, pBRight, segTangent);
04228
04229                             double crossProduct = currTangent.X * segTangent.Y - currTangent.Y *
      segTangent.X;
04230
04231                             if (crossProduct > 0 && !positive)
04232                             {
04233                                 if (lineJoin == LineJoins.Round)
04234                                 {
04235                                     tbr.Arc(p, lineWidth * 0.5, Math.Atan2(pALeft.Y - p.Y, pALeft.X -
      p.X), Math.Atan2(pBLeft.Y - p.Y, pBLeft.X - p.X));
04236                                 }
04237                                 else if (lineJoin == LineJoins.Miter)
04238                                 {
04239                                     tbr.LineTo(p3_1);
04240                                 }
04241                             }
04242                             else if (crossProduct < 0 && positive)
04243                             {
04244                                 if (lineJoin == LineJoins.Round)
04245                                 {
04246                                     tbr.Arc(p, lineWidth * 0.5, Math.Atan2(pARight.Y - p.Y, pARight.X
      - p.X), Math.Atan2(pBRight.Y - p.Y, pBRight.X - p.X));
04247                                 }
04248                                 else if (lineJoin == LineJoins.Miter)
04249                                 {
04250                                     tbr.LineTo(p3_2);
04251                                 }
04252                             }
04253                         }
04254
04255
04256                         tbr.LineTo(currPoint + sign * lineWidth * 0.5 * new Point(-segTangent.Y,
      segTangent.X));
04257
04258                         tbr.LineTo(startPoint + sign * lineWidth * 0.5 * new Point(-segTangent.Y,
      segTangent.X));
04259
04260                         currTangent = segTangent;
04261                     }
04262
04263                     currPoint = startPoint;
04264
04265
04266                     isClosed = true;
04267
04268                     started = false;
04269                 }
```

```
04270             }
04271
04272
04273             if (isClosed)
04274             {
04275                 Point segTangent = figure.GetTangentAtRelative(0);
04276
04277                 if (!double.IsNaN(currTangent.X) && !double.IsNaN(currTangent.Y) && (currTangent.X !=
      segTangent.X && currTangent.Y != segTangent.Y) && !double.IsNaN(segTangent.X) &&
      !double.IsNaN(segTangent.Y))
04278                 {
04279                     Point p = currPoint;
04280                     Point pALeft = currPoint - new Point(-currTangent.Y, currTangent.X) * lineWidth *
      0.5;
04281                     Point pARight = currPoint + new Point(-currTangent.Y, currTangent.X) * lineWidth *
      0.5;
04282
04283                     Point pBLeft = currPoint - new Point(-segTangent.Y, segTangent.X) * lineWidth *
      0.5;
04284                     Point pBRight = currPoint + new Point(-segTangent.Y, segTangent.X) * lineWidth *
      0.5;
04285
04286                     Point p3_1 = IntersectLines(pALeft, currTangent, pBLeft, segTangent);
04287                     Point p3_2 = IntersectLines(pARight, currTangent, pBRight, segTangent);
04288
04289                     double crossProduct = currTangent.X * segTangent.Y - currTangent.Y * segTangent.X;
04290
04291                     if (crossProduct > 0 && !positive)
04292                     {
04293                         if (lineJoin == LineJoins.Round)
04294                         {
04295                             double ang1 = Math.Atan2(pALeft.Y - p.Y, pALeft.X - p.X);
04296                             double ang2 = Math.Atan2(pBLeft.Y - p.Y, pBLeft.X - p.X);
04297
04298                             if (ang1 < 0)
04299                             {
04300                                 ang1 += 2 * Math.PI;
04301                             }
04302
04303                             if (ang2 < 0)
04304                             {
04305                                 ang2 += 2 * Math.PI;
04306                             }
04307
04308                             tbr.Arc(p, lineWidth * 0.5, ang1, ang2);
04309                         }
04310                         else if (lineJoin == LineJoins.Miter)
04311                         {
04312                             tbr.LineTo(p3_1);
04313                         }
04314                     }
04315                     else if (crossProduct < 0 && positive)
04316                     {
04317                         if (lineJoin == LineJoins.Round)
04318                         {
04319                             double ang1 = Math.Atan2(pARight.Y - p.Y, pARight.X - p.X);
04320                             double ang2 = Math.Atan2(pBRight.Y - p.Y, pBRight.X - p.X);
04321
04322                             if (ang1 < 0)
04323                             {
04324                                 ang1 += 2 * Math.PI;
04325                             }
04326
04327                             if (ang2 < 0)
04328                             {
04329                                 ang2 += 2 * Math.PI;
04330                             }
04331
04332                             tbr.Arc(p, lineWidth * 0.5, ang1, ang2);
04333                         }
04334                         else if (lineJoin == LineJoins.Miter)
04335                         {
04336                             tbr.LineTo(p3_2);
04337                         }
04338                     }
04339                 }
04340
04341
04342                 tbr.Close();
04343             }
04344         }
04345
04346         internal static double Tolerance = 1e-7;
04347
04348         private static GraphicsPath RemoveDuplicates(GraphicsPath path)
04349         {
04350             Point currentPoint = new Point();
```

```
04351             Point startPoint = new Point();
04352             bool started = false;
04353 
04354             GraphicsPath tbr = new GraphicsPath();
04355 
04356             GraphicsPath figure = null;
04357 
04358             for (int i = 0; i < path.Segments.Count; i++)
04359             {
04360                 if (path.Segments[i].Type == SegmentType.Move)
04361                 {
04362                     if (figure != null)
04363                     {
04364                         tbr.AddPath(figure);
04365                     }
04366 
04367                     figure = new GraphicsPath();
04368 
04369                     currentPoint = path.Segments[i].Point;
04370                     figure.MoveTo(currentPoint);
04371                     startPoint = currentPoint;
04372                     started = true;
04373                 }
04374                 else if (path.Segments[i].Type == SegmentType.Line)
04375                 {
04376                     if (figure == null)
04377                     {
04378                         figure = new GraphicsPath();
04379                     }
04380 
04381                     if (currentPoint.X != path.Segments[i].Point.X || currentPoint.Y !=
      path.Segments[i].Point.Y)
04382                     {
04383                         currentPoint = path.Segments[i].Point;
04384                         figure.LineTo(currentPoint);
04385                     }
04386 
04387                     if (!started)
04388                     {
04389                         startPoint = currentPoint;
04390                         started = true;
04391                     }
04392                 }
04393                 else if (path.Segments[i].Type == SegmentType.Arc && path.Segments[i] is ArcSegment
      arc)
04394                 {
04395                     if (figure == null)
04396                     {
04397                         figure = new GraphicsPath();
04398                     }
04399 
04400                     Point endPoint = arc.GetPointAt(currentPoint, 1);
04401 
04402                     if (arc.StartAngle != arc.EndAngle)
04403                     {
04404                         currentPoint = endPoint;
04405                         figure.Arc(arc.Points[0], arc.Radius, arc.StartAngle, arc.EndAngle);
04406                     }
04407                     else if (endPoint.X != currentPoint.X || endPoint.Y != currentPoint.Y)
04408                     {
04409                         currentPoint = endPoint;
04410                         figure.LineTo(endPoint);
04411                     }
04412 
04413                     if (!started)
04414                     {
04415                         startPoint = currentPoint;
04416                         started = true;
04417                     }
04418                 }
04419                 else if (path.Segments[i].Type == SegmentType.CubicBezier)
04420                 {
04421                     if (figure == null)
04422                     {
04423                         figure = new GraphicsPath();
04424                     }
04425 
04426                     if (!currentPoint.IsEqual(path.Segments[i].Points[0], Tolerance) ||
      !currentPoint.IsEqual(path.Segments[i].Points[1], Tolerance) ||
      !currentPoint.IsEqual(path.Segments[i].Points[2], Tolerance))
04427                     {
04428                         currentPoint = path.Segments[i].Points[2];
04429                         figure.CubicBezierTo(path.Segments[i].Points[0], path.Segments[i].Points[1],
      path.Segments[i].Points[2]);
04430                     }
04431 
04432                     if (!started)
```

```
04433                     {
04434                         startPoint = currentPoint;
04435                         started = true;
04436                     }
04437                 }
04438                 else if (path.Segments[i].Type == SegmentType.Close)
04439                 {
04440                     if (currentPoint.X != startPoint.X || currentPoint.Y != startPoint.Y)
04441                     {
04442                         currentPoint = startPoint;
04443                         figure.Close();
04444 
04445                         tbr.AddPath(figure);
04446                         figure = null;
04447                     }
04448                     else
04449                     {
04450                         if (figure.Segments[figure.Segments.Count - 1].Type == SegmentType.Line)
04451                         {
04452                             figure.Segments.RemoveAt(figure.Segments.Count - 1);
04453                         }
04454                         else if (figure.Segments[0].Type == SegmentType.Move &&
      figure.Segments[1].Type == SegmentType.Line)
04455                         {
04456                             figure.Segments[0] = new MoveSegment(figure.Segments[1].Point);
04457                             figure.Segments.RemoveAt(1);
04458                         }
04459 
04460                         currentPoint = startPoint;
04461                         figure.Close();
04462 
04463                         tbr.AddPath(figure);
04464                         figure = null;
04465                     }
04466 
04467                     started = false;
04468                 }
04469             }
04470 
04471             if (figure != null)
04472             {
04473                 tbr.AddPath(figure);
04474             }
04475 
04476             return tbr;
04477         }
04478     }
04479 }
```

8.87 ImageFormats.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
00019 using System.Collections.Generic;
00020 using System.IO;
00021 using System.IO.Compression;
00022 using System.Runtime.InteropServices;
00023 using System.Threading;
00024 
00025 namespace VectSharp
00026 {
00027 /// <summary>
00028 /// Contains methods to create animated PNG image files.
00029 /// </summary>
00030     public static class AnimatedPNG
00031     {
00032 /// <summary>
```

```
00033 /// Types of inter-frame compression.
00034 /// </summary>
00035         public enum InterframeCompression
00036         {
00037 /// <summary>
00038 /// No inter-frame compression is performed (fastest, but produces the largest files).
00039 /// </summary>
00040             None,
00041 
00042 /// <summary>
00043 /// Inter-frame compression is performed by encoding the difference between each frame and the first
      frame in the animation.
00044 /// </summary>
00045             First,
00046 
00047 /// <summary>
00048 /// Inter-frame compression is performed by encoding the difference between each frame and the
      previous frame (produces the smallest files, but it is the slowest and requires large amounts of
      memory).
00049 /// </summary>
00050             Previous
00051         }
00052 
00053 /// <summary>
00054 /// Represents an individual frame of a PNG animation.
00055 /// </summary>
00056         public class CompressedFrame :  IDisposable
00057         {
00058             internal MemoryStream compressedStream;
00059             private bool disposedValue;
00060 
00061 /// <summary>
00062 /// The duration of the frame in milliseconds.
00063 /// </summary>
00064             public double Duration { get; }
00065 
00066             internal InterframeCompression InterframeCompression { get; } =
      InterframeCompression.None;
00067 
00068 /// <summary>
00069 /// Creates a new <see cref="CompressedFrame"/> from raw pixel data in RGB or RGBA format.
00070 /// </summary>
00071 /// <param name="rawFrameData">A pointer to the raw pixel data for the frame.</param>
00072 /// <param name="width">The width of the image in pixels.</param>
00073 /// <param name="height">The height of the image in pixels.</param>
00074 /// <param name="hasAlpha">If the image data is in RGBA format, set this to <see langword="true"/>; if
      it is in RGB format, set it to <see langword="false"/>.</param>
00075 /// <param name="duration">The duration of the frame in milliseconds.</param>
00076             public unsafe CompressedFrame(DisposableIntPtr rawFrameData, int width, int height, bool
      hasAlpha, double duration)
00077             {
00078                 this.compressedStream =
      PNGEncoder.CompressAPNGFrame((byte*)rawFrameData.InternalPointer, width, height, hasAlpha,
      PNGEncoder.FilterModes.Adaptive, 1);
00079                 this.Duration = duration;
00080                 this.InterframeCompression = InterframeCompression.None;
00081             }
00082 
00083 /// <summary>
00084 /// Creates a new <see cref="CompressedFrame"/> from raw pixel data in RGB or RGBA format, applying
      inter-frame compression based on another frame.
00085 /// </summary>
00086 /// <param name="rawFrameData">A pointer to the raw pixel data for the new frame.</param>
00087 /// <param name="otherFrameData">A pointer to the raw pixel data for the frame to use as a reference.
      This can either be the first frame in the animation, or the frame immediately preceeding the current
      frame.</param>
00088 /// <param name="isFirstFrame">If <paramref name="otherFrameData"/> is the raw pixel data for the
      first frame in the animation, set this to <see langword="true"/>; if it is the immediately preceeding
      frame, set this to <see langword="false"/>.</param>
00089 /// <param name="width">The width of the image in pixels.</param>
00090 /// <param name="height">The height of the image in pixels.</param>
00091 /// <param name="hasAlpha">If the image data is in RGBA format, set this to <see langword="true"/>; if
      it is in RGB format, set it to <see langword="false"/>.</param>
00092 /// <param name="duration">The duration of the frame in milliseconds.</param>
00093             public unsafe CompressedFrame(DisposableIntPtr rawFrameData, DisposableIntPtr
      otherFrameData, bool isFirstFrame, int width, int height, bool hasAlpha, double duration)
00094             {
00095                 int pixelSize = hasAlpha ?  4 :  3;
00096 
00097                 IntPtr frameDiff = Marshal.AllocHGlobal(width * height * 4);
00098 
00099                 byte* frameDiffData = (byte*)frameDiff;
00100                 byte* previousFrame = (byte*)otherFrameData.InternalPointer;
00101                 byte* currentFrame = (byte*)rawFrameData.InternalPointer;
00102 
00103                 for (int y = 0; y < height; y++)
00104                 {
```

```
00105                     for (int x = 0; x < width; x++)
00106                     {
00107                         int pixelIndex = (y * width + x) * pixelSize;
00108 
00109                         if (previousFrame[pixelIndex] != currentFrame[pixelIndex] ||
      previousFrame[pixelIndex + 1] != currentFrame[pixelIndex + 1] ||
00110                             previousFrame[pixelIndex + 2] != currentFrame[pixelIndex + 2] || (hasAlpha
      && previousFrame[pixelIndex + 3] != currentFrame[pixelIndex + 3]))
00111                         {
00112                             frameDiffData[pixelIndex] = currentFrame[pixelIndex];
00113                             frameDiffData[pixelIndex + 1] = currentFrame[pixelIndex + 1];
00114                             frameDiffData[pixelIndex + 2] = currentFrame[pixelIndex + 2];
00115 
00116                             if (hasAlpha)
00117                             {
00118                                 frameDiffData[pixelIndex + 3] = currentFrame[pixelIndex + 3];
00119                             }
00120                             else
00121                             {
00122                                 frameDiffData[pixelIndex + 3] = 255;
00123                             }
00124                         }
00125                         else
00126                         {
00127                             frameDiffData[pixelIndex] = 0;
00128                             frameDiffData[pixelIndex + 1] = 0;
00129                             frameDiffData[pixelIndex + 2] = 0;
00130                             frameDiffData[pixelIndex + 3] = 0;
00131                         }
00132                     }
00133                 }
00134 
00135                 this.compressedStream = PNGEncoder.CompressAPNGFrame((byte*)frameDiff, width, height,
      true, PNGEncoder.FilterModes.Adaptive, 1);
00136                 this.Duration = duration;
00137 
00138                 if (isFirstFrame)
00139                 {
00140                     this.InterframeCompression = InterframeCompression.First;
00141                 }
00142                 else
00143                 {
00144                     this.InterframeCompression = InterframeCompression.Previous;
00145                 }
00146 
00147                 Marshal.FreeHGlobal(frameDiff);
00148             }
00149 
00150 /// <inheritdoc/>
00151             protected virtual void Dispose(bool disposing)
00152             {
00153                 if (!disposedValue)
00154                 {
00155                     if (disposing)
00156                     {
00157                         this.compressedStream.Dispose();
00158                     }
00159 
00160                     disposedValue = true;
00161                 }
00162             }
00163 
00164 /// <inheritdoc/>
00165             public void Dispose()
00166             {
00167                 Dispose(disposing:  true);
00168                 GC.SuppressFinalize(this);
00169             }
00170         }
00171 
00172 /// <summary>
00173 /// Create a new animated PNG image, outputting it to the specified stream.
00174 /// </summary>
00175 /// <param name="outputStream">The stream to which the animated PNG image will be written.</param>
00176 /// <param name="width">The width of the image in pixels.</param>
00177 /// <param name="height">The height of the image in pixels.</param>
00178 /// <param name="hasAlpha">If the frames of the image have an alpha channel, set this to <see
      langword="true"/>; otherwise, set it to <see langword="false"/>.</param>
00179 /// <param name="compressedFrames">The frames that will be used to create the animated PNG
      image.</param>
00180 /// <param name="repeatCount">The number of times that the animation should loop.  Set this to 0 for
      an infinitely repeating animation.</param>
00181         public static unsafe void Create(Stream outputStream, int width, int height, bool hasAlpha,
      IReadOnlyList<CompressedFrame> compressedFrames, int repeatCount)
00182         {
00183             PNGEncoder.StartAPNG(width, height, hasAlpha, outputStream, compressedFrames.Count,
      repeatCount);
```

```
00184
00185             for (int i = 0; i < compressedFrames.Count; i++)
00186             {
00187                 ushort frameDurationNumerator = (ushort)Math.Round(compressedFrames[i].Duration * 60);
00188                 ushort frameDurationDenominator = 60000;
00189                 PNGEncoder.AddPreCompressedAPNGFrame(compressedFrames[i], width, height, hasAlpha,
      outputStream, i, frameDurationNumerator, frameDurationDenominator);
00190             }
00191
00192             PNGEncoder.FinishAPNG(outputStream);
00193         }
00194     }
00195
00196
00197
00198     internal static class PNGEncoder
00199     {
00200         internal unsafe static void SavePNG(byte* image, int width, int height, bool hasAlpha, Stream
      fs, FilterModes filter, int threadCount = 0)
00201         {
00202             if (threadCount == 0)
00203             {
00204                 threadCount = filter == FilterModes.Adaptive ?  Math.Max(1, Math.Min(width / 600,
      Environment.ProcessorCount - 2)) :  1;
00205             }
00206
00207             //Header
00208             fs.Write(new byte[] { 0x89, 0x50, 0x4E, 0x47, 0x0D, 0x0A, 0x1A, 0x0A }, 0, 8);
00209
00210             //IHDR chunk
00211             fs.WriteInt(13);
00212             using (MemoryStream ihdr = new MemoryStream(13))
00213             {
00214                 ihdr.WriteASCIIString("IHDR");
00215                 ihdr.WriteInt(width);
00216                 ihdr.WriteInt(height);
00217                 ihdr.WriteByte(8); //Bit depth
00218
00219                 if (hasAlpha)
00220                 {
00221                     ihdr.WriteByte(6); //Colour type
00222                 }
00223                 else
00224                 {
00225                     ihdr.WriteByte(2); //Colour type
00226                 }
00227
00228                 ihdr.WriteByte(0); //Compression method
00229                 ihdr.WriteByte(0); //Filter method
00230                 ihdr.WriteByte(0); //Interlace
00231
00232                 ihdr.Seek(0, SeekOrigin.Begin);
00233                 ihdr.CopyTo(fs);
00234
00235                 fs.WriteUInt(CRC32.ComputeCRC(ihdr));
00236             }
00237
00238             //IDAT chunk
00239             IntPtr filteredImage;
00240
00241             if (threadCount > 1)
00242             {
00243                 filteredImage = FilterImageData(image, width, height, hasAlpha ?  4 :  3, filter,
      threadCount);
00244             }
00245             else
00246             {
00247                 filteredImage = FilterImageData(image, width, height, hasAlpha ?  4 :  3, filter);
00248             }
00249
00250             using (MemoryStream compressedImage = StreamUtils.ZLibCompress(filteredImage, height *
      (width * (hasAlpha ?  4 :  3) + 1)))
00251             {
00252                 compressedImage.Seek(0, SeekOrigin.Begin);
00253                 fs.WriteUInt((uint)compressedImage.Length);
00254                 fs.WriteASCIIString("IDAT");
00255                 compressedImage.Seek(0, SeekOrigin.Begin);
00256                 compressedImage.CopyTo(fs);
00257
00258                 fs.WriteUInt(CRC32.ComputeCRC(compressedImage.GetBuffer(),
      (int)compressedImage.Length, new byte[] { 73, 68, 65, 84 }));
00259             }
00260
00261             Marshal.FreeHGlobal(filteredImage);
00262
00263             //IEND chunk
00264             fs.WriteInt(0);
```

```
00265             fs.WriteASCIIString("IEND");
00266             fs.Write(new byte[] { 0xAE, 0x42, 0x60, 0x82 }, 0, 4);
00267 
00268         }
00269 
00270         public unsafe static void StartAPNG(int width, int height, bool hasAlpha, Stream fs, int
      frameCount, int repeatCount)
00271         {
00272             //Header
00273             fs.Write(new byte[] { 0x89, 0x50, 0x4E, 0x47, 0x0D, 0x0A, 0x1A, 0x0A }, 0, 8);
00274 
00275             //IHDR chunk
00276             fs.WriteInt(13);
00277             using (MemoryStream ihdr = new MemoryStream(17))
00278             {
00279                 ihdr.WriteASCIIString("IHDR");
00280                 ihdr.WriteInt(width);
00281                 ihdr.WriteInt(height);
00282                 ihdr.WriteByte(8); //Bit depth
00283 
00284                 if (hasAlpha)
00285                 {
00286                     ihdr.WriteByte(6); //Colour type
00287                 }
00288                 else
00289                 {
00290                     ihdr.WriteByte(2); //Colour type
00291                 }
00292 
00293                 ihdr.WriteByte(0); //Compression method
00294                 ihdr.WriteByte(0); //Filter method
00295                 ihdr.WriteByte(0); //Interlace
00296 
00297                 ihdr.Seek(0, SeekOrigin.Begin);
00298                 ihdr.CopyTo(fs);
00299 
00300                 fs.WriteUInt(CRC32.ComputeCRC(ihdr));
00301             }
00302 
00303             //acTL chunk
00304             fs.WriteInt(8);
00305             using (MemoryStream actl = new MemoryStream(12))
00306             {
00307                 actl.WriteASCIIString("acTL");
00308                 actl.WriteInt(frameCount);
00309                 actl.WriteInt(repeatCount);
00310 
00311                 actl.Seek(0, SeekOrigin.Begin);
00312                 actl.CopyTo(fs);
00313 
00314                 fs.WriteUInt(CRC32.ComputeCRC(actl));
00315             }
00316         }
00317 
00318         internal unsafe static void AddAPNGFrame(byte* image, int width, int height, bool hasAlpha,
      Stream fs, FilterModes filter, int frameNumber, ushort frameDelayNumerator, ushort
      frameDelayDenominator, int threadCount = 0)
00319         {
00320             if (threadCount == 0)
00321             {
00322                 threadCount = filter == FilterModes.Adaptive ?  Math.Max(1, Math.Min(width / 600,
      Environment.ProcessorCount - 2)) :  1;
00323             }
00324 
00325             int fcTLIndex = frameNumber == 0 ?  0 :  ((frameNumber - 1) * 2 + 1);
00326             int fDATIndex = frameNumber == 0 ?  0 :  (frameNumber * 2);
00327 
00328             //fcTL chunk
00329             fs.WriteInt(26);
00330             using (MemoryStream fctl = new MemoryStream(30))
00331             {
00332                 fctl.WriteASCIIString("fcTL");
00333                 fctl.WriteInt(fcTLIndex);
00334                 fctl.WriteInt(width);
00335                 fctl.WriteInt(height);
00336                 fctl.WriteInt(0);
00337                 fctl.WriteInt(0);
00338                 fctl.WriteUShort(frameDelayNumerator);
00339                 fctl.WriteUShort(frameDelayDenominator);
00340                 fctl.WriteByte(1);
00341                 fctl.WriteByte(0);
00342 
00343                 fctl.Seek(0, SeekOrigin.Begin);
00344                 fctl.CopyTo(fs);
00345 
00346                 fs.WriteUInt(CRC32.ComputeCRC(fctl));
00347             }
```

```
00348
00349             //fDAT chunk
00350             IntPtr filteredImage;
00351
00352             if (threadCount > 1)
00353             {
00354                 filteredImage = FilterImageData(image, width, height, hasAlpha ?  4 :  3, filter,
      threadCount);
00355             }
00356             else
00357             {
00358                 filteredImage = FilterImageData(image, width, height, hasAlpha ?  4 :  3, filter);
00359             }
00360
00361             if (frameNumber > 0)
00362             {
00363                 using (MemoryStream compressedImage = StreamUtils.ZLibCompress(filteredImage, height *
      (width * (hasAlpha ?  4 :  3) + 1)))
00364                 {
00365                     fs.WriteUInt((uint)compressedImage.Length + 4);
00366
00367                     fs.WriteASCIIString("fdAT");
00368                     fs.WriteInt(fDATIndex);
00369
00370                     compressedImage.Seek(0, SeekOrigin.Begin);
00371                     compressedImage.CopyTo(fs);
00372
00373                     fs.WriteUInt(CRC32.ComputeCRC(compressedImage.GetBuffer(),
      (int)compressedImage.Length, new byte[] { 102, 100, 65, 84, (byte)(fDATIndex » 24), (byte)((fDATIndex
      » 16) & 255), (byte)((fDATIndex » 8) & 255), (byte)(fDATIndex & 255) }));
00374                     //fs.WriteUInt(CRC32.ComputeCRC(chunkStream.GetBuffer(), (int)chunkStream.Length,
      new byte[] { 73, 68, 65, 84 }));
00375                     //fs.WriteUInt(CRC32.ComputeCRC(chunkStream));
00376                 }
00377             }
00378             else
00379             {
00380                 using (MemoryStream compressedImage = StreamUtils.ZLibCompress(filteredImage, height *
      (width * (hasAlpha ?  4 :  3) + 1)))
00381                 {
00382                     compressedImage.Seek(0, SeekOrigin.Begin);
00383                     fs.WriteUInt((uint)compressedImage.Length);
00384                     fs.WriteASCIIString("IDAT");
00385                     compressedImage.Seek(0, SeekOrigin.Begin);
00386                     compressedImage.CopyTo(fs);
00387
00388                     fs.WriteUInt(CRC32.ComputeCRC(compressedImage.GetBuffer(),
      (int)compressedImage.Length, new byte[] { 73, 68, 65, 84 }));
00389                 }
00390             }
00391
00392             Marshal.FreeHGlobal(filteredImage);
00393         }
00394
00395         internal static unsafe MemoryStream CompressAPNGFrame(byte* image, int width, int height, bool
      hasAlpha, FilterModes filter, int threadCount)
00396         {
00397             //fDAT chunk
00398             IntPtr filteredImage;
00399
00400             if (threadCount > 1)
00401             {
00402                 filteredImage = FilterImageData(image, width, height, hasAlpha ?  4 :  3, filter,
      threadCount);
00403             }
00404             else
00405             {
00406                 filteredImage = FilterImageData(image, width, height, hasAlpha ?  4 :  3, filter);
00407             }
00408
00409             MemoryStream compressedImage = StreamUtils.ZLibCompress(filteredImage, height * (width *
      (hasAlpha ?  4 :  3) + 1));
00410
00411             Marshal.FreeHGlobal(filteredImage);
00412
00413             return compressedImage;
00414         }
00415
00416         internal static void AddPreCompressedAPNGFrame(AnimatedPNG.CompressedFrame compressedFrame,
      int width, int height, bool hasAlpha, Stream fs, int frameNumber, ushort frameDelayNumerator, ushort
      frameDelayDenominator)
00417         {
00418             int fcTLIndex = frameNumber == 0 ?  0 :  ((frameNumber - 1) * 2 + 1);
00419             int fDATIndex = frameNumber == 0 ?  0 :  (frameNumber * 2);
00420
00421             //fcTL chunk
00422             fs.WriteInt(26);
```

```
00423             using (MemoryStream fctl = new MemoryStream(30))
00424             {
00425                 fctl.WriteASCIIString("fcTL");
00426                 fctl.WriteInt(fcTLIndex);
00427                 fctl.WriteInt(width);
00428                 fctl.WriteInt(height);
00429                 fctl.WriteInt(0);
00430                 fctl.WriteInt(0);
00431                 fctl.WriteUShort(frameDelayNumerator);
00432                 fctl.WriteUShort(frameDelayDenominator);
00433 
00434                 if (compressedFrame.InterframeCompression == AnimatedPNG.InterframeCompression.None ||
      frameNumber == 0)
00435                 {
00436                     fctl.WriteByte(0);
00437                     fctl.WriteByte(0);
00438                 }
00439                 else if (compressedFrame.InterframeCompression ==
      AnimatedPNG.InterframeCompression.First)
00440                 {
00441                     fctl.WriteByte(2);
00442                     fctl.WriteByte(1);
00443                 }
00444                 else if (compressedFrame.InterframeCompression ==
      AnimatedPNG.InterframeCompression.Previous)
00445                 {
00446                     fctl.WriteByte(0);
00447                     fctl.WriteByte(1);
00448                 }
00449 
00450                 fctl.Seek(0, SeekOrigin.Begin);
00451                 fctl.CopyTo(fs);
00452 
00453                 fs.WriteUInt(CRC32.ComputeCRC(fctl));
00454             }
00455 
00456             MemoryStream compressedImage = compressedFrame.compressedStream;
00457 
00458             if (frameNumber > 0)
00459             {
00460                 fs.WriteUInt((uint)compressedImage.Length + 4);
00461 
00462                 fs.WriteASCIIString("fdAT");
00463                 fs.WriteInt(fDATIndex);
00464 
00465                 compressedImage.Seek(0, SeekOrigin.Begin);
00466                 compressedImage.CopyTo(fs);
00467 
00468                 fs.WriteUInt(CRC32.ComputeCRC(compressedImage.GetBuffer(),
      (int)compressedImage.Length, new byte[] { 102, 100, 65, 84, (byte)(fDATIndex » 24), (byte)((fDATIndex
      » 16) & 255), (byte)((fDATIndex » 8) & 255), (byte)(fDATIndex & 255) }));
00469             }
00470             else
00471             {
00472                 compressedImage.Seek(0, SeekOrigin.Begin);
00473                 fs.WriteUInt((uint)compressedImage.Length);
00474                 fs.WriteASCIIString("IDAT");
00475                 compressedImage.Seek(0, SeekOrigin.Begin);
00476                 compressedImage.CopyTo(fs);
00477 
00478                 fs.WriteUInt(CRC32.ComputeCRC(compressedImage.GetBuffer(),
      (int)compressedImage.Length, new byte[] { 73, 68, 65, 84 }));
00479             }
00480         }
00481 
00482         internal unsafe static void FinishAPNG(Stream fs)
00483         {
00484             //IEND chunk
00485             fs.WriteInt(0);
00486             fs.WriteASCIIString("IEND");
00487             fs.Write(new byte[] { 0xAE, 0x42, 0x60, 0x82 }, 0, 4);
00488         }
00489 
00490         internal enum FilterModes
00491         {
00492             None = 0,
00493             Sub = 1,
00494             Up = 2,
00495             Average = 3,
00496             Paeth = 4,
00497             Adaptive = -1
00498         }
00499 
00500         internal unsafe static void FilterRow(byte* image, int width, int pixelSize, int stride, int
      y, FilterModes filter, byte* destinationImage, byte* tempBuffer)
00501         {
00502             int startIndex = y * stride;
```

```
00503             int destinationIndex = y * (stride + 1);
00504
00505             if (filter == FilterModes.None)
00506             {
00507                 destinationImage[destinationIndex++] = 0;
00508                 for (int x = 0; x < width; x++)
00509                 {
00510                     destinationImage[destinationIndex++] = image[startIndex++];
00511                     destinationImage[destinationIndex++] = image[startIndex++];
00512                     destinationImage[destinationIndex++] = image[startIndex++];
00513
00514                     if (pixelSize == 4)
00515                     {
00516                         destinationImage[destinationIndex++] = image[startIndex++];
00517                     }
00518
00519                 }
00520             }
00521             else if (filter == FilterModes.Sub)
00522             {
00523                 destinationImage[destinationIndex++] = 1;
00524                 for (int x = 0; x < width; x++)
00525                 {
00526                     if (x > 0)
00527                     {
00528                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - pixelSize]);
00529                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - pixelSize]);
00530                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - pixelSize]);
00531                         if (pixelSize == 4)
00532                         {
00533                             destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - pixelSize]);
00534                         }
00535                     }
00536                     else
00537                     {
00538                         destinationImage[destinationIndex++] = image[startIndex++];
00539                         destinationImage[destinationIndex++] = image[startIndex++];
00540                         destinationImage[destinationIndex++] = image[startIndex++];
00541                         if (pixelSize == 4)
00542                         {
00543                             destinationImage[destinationIndex++] = image[startIndex++];
00544                         }
00545                     }
00546                 }
00547             }
00548             else if (filter == FilterModes.Up)
00549             {
00550                 destinationImage[destinationIndex++] = 2;
00551                 for (int x = 0; x < width; x++)
00552                 {
00553                     if (y > 0)
00554                     {
00555                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - stride]);
00556                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - stride]);
00557                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - stride]);
00558                         if (pixelSize == 4)
00559                         {
00560                             destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - stride]);
00561                         }
00562                     }
00563                     else
00564                     {
00565                         destinationImage[destinationIndex++] = image[startIndex++];
00566                         destinationImage[destinationIndex++] = image[startIndex++];
00567                         destinationImage[destinationIndex++] = image[startIndex++];
00568                         if (pixelSize == 4)
00569                         {
00570                             destinationImage[destinationIndex++] = image[startIndex++];
00571                         }
00572                     }
00573                 }
00574             }
00575             else if (filter == FilterModes.Average)
00576             {
00577                 destinationImage[destinationIndex++] = 3;
00578                 for (int x = 0; x < width; x++)
00579                 {
00580                     if (x > 0 && y > 0)
00581                     {
```

```
00582                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      (image[startIndex - pixelSize] + image[startIndex++ - stride]) / 2);
00583                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      (image[startIndex - pixelSize] + image[startIndex++ - stride]) / 2);
00584                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      (image[startIndex - pixelSize] + image[startIndex++ - stride]) / 2);
00585                         if (pixelSize == 4)
00586                         {
00587                             destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      (image[startIndex - pixelSize] + image[startIndex++ - stride]) / 2);
00588                         }
00589                     }
00590                     else if (x > 0 && y == 0)
00591                     {
00592                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - pixelSize] / 2);
00593                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - pixelSize] / 2);
00594                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - pixelSize] / 2);
00595                         if (pixelSize == 4)
00596                         {
00597                             destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - pixelSize] / 2);
00598                         }
00599                     }
00600                     else if (x == 0 && y > 0)
00601                     {
00602                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - stride] / 2);
00603                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - stride] / 2);
00604                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - stride] / 2);
00605                         if (pixelSize == 4)
00606                         {
00607                             destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - stride] / 2);
00608                         }
00609                     }
00610                     else
00611                     {
00612                         destinationImage[destinationIndex++] = image[startIndex++];
00613                         destinationImage[destinationIndex++] = image[startIndex++];
00614                         destinationImage[destinationIndex++] = image[startIndex++];
00615                         if (pixelSize == 4)
00616                         {
00617                             destinationImage[destinationIndex++] = image[startIndex++];
00618                         }
00619                     }
00620                 }
00621             }
00622             else if (filter == FilterModes.Paeth)
00623             {
00624                 destinationImage[destinationIndex++] = 4;
00625                 for (int x = 0; x < width; x++)
00626                 {
00627                     if (x > 0 && y > 0)
00628                     {
00629                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      PaethPredictor(image[startIndex - pixelSize], image[startIndex - stride], image[startIndex++ -
      pixelSize - stride]));
00630                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      PaethPredictor(image[startIndex - pixelSize], image[startIndex - stride], image[startIndex++ -
      pixelSize - stride]));
00631                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      PaethPredictor(image[startIndex - pixelSize], image[startIndex - stride], image[startIndex++ -
      pixelSize - stride]));
00632                         if (pixelSize == 4)
00633                         {
00634                             destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      PaethPredictor(image[startIndex - pixelSize], image[startIndex - stride], image[startIndex++ -
      pixelSize - stride]));
00635                         }
00636                     }
00637                     else if (x > 0 && y == 0)
00638                     {
00639                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - pixelSize]);
00640                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - pixelSize]);
00641                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - pixelSize]);
00642                         if (pixelSize == 4)
00643                         {
00644                             destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - pixelSize]);
```

```
00645                         }
00646                     }
00647                     else if (x == 0 && y > 0)
00648                     {
00649                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - stride]);
00650                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - stride]);
00651                         destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - stride]);
00652                         if (pixelSize == 4)
00653                         {
00654                             destinationImage[destinationIndex++] = (byte)(image[startIndex] -
      image[startIndex++ - stride]);
00655                         }
00656                     }
00657                     else
00658                     {
00659                         destinationImage[destinationIndex++] = image[startIndex++];
00660                         destinationImage[destinationIndex++] = image[startIndex++];
00661                         destinationImage[destinationIndex++] = image[startIndex++];
00662                         if (pixelSize == 4)
00663                         {
00664                             destinationImage[destinationIndex++] = image[startIndex++];
00665                         }
00666                     }
00667                 }
00668             }
00669             else if (filter == FilterModes.Adaptive)
00670             {
00671                 int tempIndex = 0;
00672 
00673                 int none = 0;
00674                 int sub = 0;
00675                 int up = 0;
00676                 int average = 0;
00677                 int paeth = 0;
00678 
00679                 for (int x = 0; x < width; x++)
00680                 {
00681                     for (int i = 0; i < pixelSize; i++)
00682                     {
00683                         //none
00684                         tempBuffer[tempIndex] = image[startIndex];
00685 
00686                         //sub
00687                         if (x > 0)
00688                         {
00689                             tempBuffer[tempIndex + stride] = (byte)(image[startIndex] -
      image[startIndex - pixelSize]);
00690                         }
00691                         else
00692                         {
00693                             tempBuffer[tempIndex + stride] = image[startIndex];
00694                         }
00695 
00696                         //up
00697                         if (y > 0)
00698                         {
00699                             tempBuffer[tempIndex + stride * 2] = (byte)(image[startIndex] -
      image[startIndex - stride]);
00700                         }
00701                         else
00702                         {
00703                             tempBuffer[tempIndex + stride * 2] = image[startIndex];
00704                         }
00705 
00706                         //average
00707                         if (x > 0 && y > 0)
00708                         {
00709                             tempBuffer[tempIndex + stride * 3] = (byte)(image[startIndex] -
      (image[startIndex - pixelSize] + image[startIndex - stride]) / 2);
00710                         }
00711                         else if (x > 0 && y == 0)
00712                         {
00713                             tempBuffer[tempIndex + stride * 3] = (byte)(image[startIndex] -
      image[startIndex - pixelSize] / 2);
00714                         }
00715                         else if (x == 0 && y > 0)
00716                         {
00717                             tempBuffer[tempIndex + stride * 3] = (byte)(image[startIndex] -
      image[startIndex - stride] / 2);
00718                         }
00719                         else
00720                         {
00721                             tempBuffer[tempIndex + stride * 3] = image[startIndex];
00722                         }
```

```
00723
00724                       //paeth
00725                       if (x > 0 && y > 0)
00726                       {
00727                           tempBuffer[tempIndex + stride * 4] = (byte)(image[startIndex] -
      PaethPredictor(image[startIndex - pixelSize], image[startIndex - stride], image[startIndex - pixelSize
      - stride]));
00728                       }
00729                       else if (x > 0 && y == 0)
00730                       {
00731                           tempBuffer[tempIndex + stride * 4] = (byte)(image[startIndex] -
      image[startIndex - pixelSize]);
00732                       }
00733                       else if (x == 0 && y > 0)
00734                       {
00735                           tempBuffer[tempIndex + stride * 4] = (byte)(image[startIndex] -
      image[startIndex - stride]);
00736                       }
00737                       else
00738                       {
00739                           tempBuffer[tempIndex + stride * 4] = image[startIndex];
00740                       }
00741
00742                       none += tempBuffer[tempIndex];
00743                       sub += tempBuffer[tempIndex + stride];
00744                       up += tempBuffer[tempIndex + stride * 2];
00745                       average += tempBuffer[tempIndex + stride * 3];
00746                       paeth += tempBuffer[tempIndex + stride * 4];
00747                       tempIndex++;
00748                       startIndex++;
00749                   }
00750               }
00751
00752               if (paeth <= average && paeth <= up && paeth <= sub && paeth <= none)
00753               {
00754                   destinationImage[destinationIndex++] = 4;
00755                   tempIndex = stride * 4;
00756                   for (int i = 0; i < stride; i++)
00757                   {
00758                       destinationImage[destinationIndex++] = tempBuffer[tempIndex++];
00759                   }
00760               }
00761               else if (average <= up && average <= sub && average <= none)
00762               {
00763                   destinationImage[destinationIndex++] = 3;
00764                   tempIndex = stride * 3;
00765                   for (int i = 0; i < stride; i++)
00766                   {
00767                       destinationImage[destinationIndex++] = tempBuffer[tempIndex++];
00768                   }
00769               }
00770               else if (up <= sub && up <= none)
00771               {
00772                   destinationImage[destinationIndex++] = 2;
00773                   tempIndex = stride * 2;
00774                   for (int i = 0; i < stride; i++)
00775                   {
00776                       destinationImage[destinationIndex++] = tempBuffer[tempIndex++];
00777                   }
00778               }
00779               else if (sub <= none)
00780               {
00781                   destinationImage[destinationIndex++] = 1;
00782                   tempIndex = stride;
00783                   for (int i = 0; i < stride; i++)
00784                   {
00785                       destinationImage[destinationIndex++] = tempBuffer[tempIndex++];
00786                   }
00787               }
00788               else
00789               {
00790                   destinationImage[destinationIndex++] = 0;
00791                   tempIndex = 0;
00792                   for (int i = 0; i < stride; i++)
00793                   {
00794                       destinationImage[destinationIndex++] = tempBuffer[tempIndex++];
00795                   }
00796               }
00797           }
00798       }
00799
00800       internal unsafe static IntPtr FilterImageData(byte* image, int width, int height, int
      pixelSize, FilterModes filter)
00801       {
00802           IntPtr filteredMemory = Marshal.AllocHGlobal(height * (1 + width * pixelSize));
00803
00804           IntPtr tempBuffer;
```

```
00805
00806             if (filter == FilterModes.Adaptive)
00807             {
00808                 tempBuffer = Marshal.AllocHGlobal(5 * width * pixelSize);
00809             }
00810             else
00811             {
00812                 tempBuffer = IntPtr.Zero;
00813             }
00814
00815             int stride = width * pixelSize;
00816
00817             byte* filteredPointer = (byte*)filteredMemory;
00818
00819             byte* tempPointer = (byte*)tempBuffer;
00820
00821             for (int y = 0; y < height; y++)
00822             {
00823                 FilterRow(image, width, pixelSize, stride, y, filter, filteredPointer, tempPointer);
00824             }
00825
00826             Marshal.FreeHGlobal(tempBuffer);
00827
00828             return filteredMemory;
00829         }
00830
00831         internal unsafe static IntPtr FilterImageData(byte* image, int width, int height, int
      pixelSize, FilterModes filter, int threadCount)
00832         {
00833             threadCount = Math.Min(threadCount, height);
00834
00835             IntPtr filteredMemory = Marshal.AllocHGlobal(height * (1 + width * pixelSize));
00836
00837             int stride = width * pixelSize;
00838
00839             byte* filteredPointer = (byte*)filteredMemory;
00840
00841             IntPtr[] tempBuffers = new IntPtr[threadCount];
00842
00843             Thread[] threads = new Thread[threadCount];
00844             int[] rowsByThread = new int[threadCount];
00845             EventWaitHandle[] signalsFromThreads = new EventWaitHandle[threadCount];
00846             EventWaitHandle[] signalsToThreads = new EventWaitHandle[threadCount];
00847
00848             int firstNeededRow = threadCount;
00849
00850             for (int i = 0; i < threadCount; i++)
00851             {
00852                 rowsByThread[i] = i;
00853                 signalsFromThreads[i] = new EventWaitHandle(false, EventResetMode.ManualReset);
00854                 signalsToThreads[i] = new EventWaitHandle(false, EventResetMode.ManualReset);
00855
00856                 if (filter == FilterModes.Adaptive)
00857                 {
00858                     tempBuffers[i] = Marshal.AllocHGlobal(5 * width * pixelSize);
00859                 }
00860                 else
00861                 {
00862                     tempBuffers[i] = IntPtr.Zero;
00863                 }
00864
00865                 byte* tempPointer = (byte*)tempBuffers[i];
00866
00867                 int threadIndex = i;
00868
00869                 threads[i] = new Thread(() =>
00870                 {
00871                     while (rowsByThread[threadIndex] >= 0)
00872                     {
00873                         FilterRow(image, width, pixelSize, stride, rowsByThread[threadIndex], filter,
      filteredPointer, tempPointer);
00874                         EventWaitHandle.SignalAndWait(signalsFromThreads[threadIndex],
      signalsToThreads[threadIndex]);
00875                         signalsToThreads[threadIndex].Reset();
00876                     }
00877                 });
00878             }
00879
00880             int finished = 0;
00881
00882             for (int i = 0; i < threadCount; i++)
00883             {
00884                 threads[i].Start();
00885             }
00886
00887             while (finished < threadCount)
00888             {
```

```
00889                 int threadInd = EventWaitHandle.WaitAny(signalsFromThreads);
00890
00891                 signalsFromThreads[threadInd].Reset();
00892
00893                 if (firstNeededRow < height)
00894                 {
00895                     rowsByThread[threadInd] = firstNeededRow;
00896                     firstNeededRow++;
00897                     signalsToThreads[threadInd].Set();
00898                 }
00899                 else
00900                 {
00901                     rowsByThread[threadInd] = -1;
00902                     signalsToThreads[threadInd].Set();
00903                     finished++;
00904                 }
00905             }
00906
00907             for (int i = 0; i < threadCount; i++)
00908             {
00909                 if (filter == FilterModes.Adaptive)
00910                 {
00911                     Marshal.FreeHGlobal(tempBuffers[i]);
00912                 }
00913             }
00914
00915             return filteredMemory;
00916         }
00917
00918         //Based on http://www.libpng.org/pub/png/spec/1.2/PNG-Filters.html
00919         internal static byte PaethPredictor(byte a, byte b, byte c)
00920         {
00921             int p = (int)a + (int)b - (int)c;
00922             int pa = Math.Abs(p - (int)a);
00923             int pb = Math.Abs(p - (int)b);
00924             int pc = Math.Abs(p - (int)c);
00925
00926             if (pa <= pb && pa <= pc)
00927             {
00928                 return a;
00929             }
00930             else if (pb <= pc)
00931             {
00932                 return b;
00933             }
00934             else
00935             {
00936                 return c;
00937             }
00938         }
00939     }
00940
00941     //Derived from http://www.libpng.org/pub/png/spec/1.2/PNG-CRCAppendix.html
00942     internal static class CRC32
00943     {
00944         /* Table of CRCs of all 8-bit messages.  */
00945         private static readonly uint[] crc_table = new uint[256];
00946
00947         /* Flag:  has the table been computed?  Initially false.  */
00948         private static bool crc_table_computed = false;
00949
00950         /* Make the table for a fast CRC. */
00951         private static void MakeCRCtable()
00952         {
00953             uint c;
00954             int n, k;
00955
00956             for (n = 0; n < 256; n++)
00957             {
00958                 c = (uint)n;
00959                 for (k = 0; k < 8; k++)
00960                 {
00961                     if ((c & 1) != 0)
00962                     {
00963                         c = 0xedb88320 ^ (c » 1);
00964                     }
00965                     else
00966                     {
00967                         c »= 1;
00968                     }
00969                 }
00970                 crc_table[n] = c;
00971             }
00972             crc_table_computed = true;
00973         }
00974
00975         /* Update a running CRC with the bytes buf[0..len-1]--the CRC
```

```
00976 should be initialized to all 1's, and the transmitted value
00977 is the 1's complement of the final running CRC (see the
00978 crc() routine below)).  */
00979
00980         private static uint UpdateCRC(uint crc, Stream buf)
00981         {
00982             uint c = crc;
00983             int n;
00984
00985             if (!crc_table_computed)
00986             {
00987                 MakeCRCtable();
00988             }
00989
00990             buf.Seek(0, SeekOrigin.Begin);
00991
00992             for (n = 0; n < buf.Length; n++)
00993             {
00994                 c = crc_table[(c ^ (byte)buf.ReadByte()) & 0xff] ^ (c » 8);
00995             }
00996             return c;
00997         }
00998
00999         private static uint UpdateCRC(uint crc, byte[] buf, int length)
01000         {
01001             uint c = crc;
01002             int n;
01003
01004             if (!crc_table_computed)
01005             {
01006                 MakeCRCtable();
01007             }
01008
01009             for (n = 0; n < length; n++)
01010             {
01011                 c = crc_table[(c ^ buf[n]) & 0xff] ^ (c » 8);
01012             }
01013             return c;
01014         }
01015
01016         /* Return the CRC of the bytes buf[0..len-1].  */
01017         public static uint ComputeCRC(Stream buf)
01018         {
01019             return UpdateCRC(0xffffffff, buf) ^ 0xffffffff;
01020         }
01021
01022         public static uint ComputeCRC(byte[] buf, int length, byte[] prefix)
01023         {
01024             uint temp = UpdateCRC(0xffffffff, prefix, prefix.Length);
01025             return UpdateCRC(temp, buf, length) ^ 0xffffffff;
01026         }
01027     }
01028
01029     internal static class StreamUtils
01030     {
01031         public static void WriteASCIIString(this Stream sr, string val)
01032         {
01033             foreach (char c in val)
01034             {
01035                 sr.WriteByte((byte)c);
01036             }
01037         }
01038
01039         internal static MemoryStream ZLibCompress(IntPtr memoryAddress, int memoryLength)
01040         {
01041             MemoryStream compressedStream = new MemoryStream();
01042             compressedStream.Write(new byte[] { 0x78, 0x01 }, 0, 2);
01043
01044             byte[] buffer = new byte[1024];
01045
01046             AdlerHolder adlerHolder = new AdlerHolder();
01047
01048             using (DeflateStream deflate = new DeflateStream(compressedStream,
      CompressionLevel.Optimal, true))
01049             {
01050                 while (memoryLength > 0)
01051                 {
01052                     int currentBytes = Math.Min(memoryLength, 1024);
01053                     Marshal.Copy(memoryAddress, buffer, 0, currentBytes);
01054                     deflate.Write(buffer, 0, currentBytes);
01055                     memoryLength -= currentBytes;
01056                     memoryAddress = IntPtr.Add(memoryAddress, currentBytes);
01057                     Adler32(buffer, currentBytes, adlerHolder);
01058                 }
01059             }
01060
01061
```

```
01062             uint checksum = (adlerHolder.s2 << 16) + adlerHolder.s1;
01063
01064             compressedStream.Write(new byte[] { (byte)((checksum >> 24) & 255), (byte)((checksum >> 16)
      & 255), (byte)((checksum >> 8) & 255), (byte)(checksum & 255) }, 0, 4);
01065
01066             compressedStream.Seek(0, SeekOrigin.Begin);
01067
01068             return compressedStream;
01069         }
01070
01071         internal static uint Adler32(Stream contentStream)
01072         {
01073             uint s1 = 1;
01074             uint s2 = 0;
01075
01076             int readByte;
01077
01078             while ((readByte = contentStream.ReadByte()) >= 0)
01079             {
01080                 s1 = (s1 + (byte)readByte) % 65521U;
01081                 s2 = (s2 + s1) % 65521U;
01082             }
01083
01084             return (s2 << 16) + s1;
01085         }
01086
01087
01088         internal class AdlerHolder
01089         {
01090             public uint s1;
01091             public uint s2;
01092
01093             public AdlerHolder()
01094             {
01095                 s1 = 1;
01096                 s2 = 0;
01097             }
01098         }
01099
01100         internal static void Adler32(byte[] buffer, int length, AdlerHolder holder)
01101         {
01102             for (int i = 0; i < length; i++)
01103             {
01104                 holder.s1 = (holder.s1 + buffer[i]) % 65521U;
01105                 holder.s2 = (holder.s2 + holder.s1) % 65521U;
01106             }
01107         }
01108     }
01109 }
```

8.88 Point.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections;
00020 using System.Collections.Generic;
00021 using System.Diagnostics;
00022 using System.Diagnostics.Contracts;
00023 using System.Linq;
00024
00025 namespace VectSharp
00026 {
00027 /// <summary>
00028 /// Represents a point relative to an origin in the top-left corner.
00029 /// </summary>
00030     [DebuggerDisplay("\\{ X: {X}, Y: {Y} \\}")]
00031     public struct Point :  IReadOnlyList<double>
```

```
00032     {
00033 /// <summary>
00034 /// Horizontal (x) coordinate, measured to the right of the origin.
00035 /// </summary>
00036         public double X;
00037 
00038 /// <summary>
00039 /// Vertical (y) coordinate, measured to the bottom of the origin.
00040 /// </summary>
00041         public double Y;
00042 
00043 /// <inheritdoc/>
00044         public int Count => 2;
00045 
00046 /// <inheritdoc/>
00047         public double this[int index]
00048         {
00049             get
00050             {
00051                 if (index == 0)
00052                 {
00053                     return this.X;
00054                 }
00055                 else if (index == 1)
00056                 {
00057                     return this.Y;
00058                 }
00059                 else
00060                 {
00061                     throw new IndexOutOfRangeException();
00062                 }
00063             }
00064         }
00065 
00066 /// <summary>
00067 /// Create a new <see cref="Point"/>.
00068 /// </summary>
00069 /// <param name="x">The horizontal (x) coordinate.</param>
00070 /// <param name="y">The vertical (y) coordinate.</param>
00071         public Point(double x, double y)
00072         {
00073             X = x;
00074             Y = y;
00075         }
00076 
00077 /// <summary>
00078 /// Computes the modulus of the vector represented by the <see cref="Point"/>.
00079 /// </summary>
00080 /// <returns>The modulus of the vector represented by the <see cref="Point"/>.</returns>
00081         public double Modulus()
00082         {
00083             return Math.Sqrt(X * X + Y * Y);
00084         }
00085 
00086 /// <summary>
00087 /// Normalises a <see cref="Point"/>.
00088 /// </summary>
00089 /// <returns>The normalised <see cref="Point"/>.</returns>
00090         [Pure]
00091         public Point Normalize()
00092         {
00093             double mod = Modulus();
00094             return new Point(X / mod, Y / mod);
00095         }
00096 
00097 /// <summary>
00098 /// Checks whether this <see cref="Point"/> is equal to another <see cref="Point"/>, up to a specified
      tolerance.
00099 /// </summary>
00100 /// <param name="p2">The <see cref="Point"/> to compare.</param>
00101 /// <param name="tolerance">The tolerance threshold.</param>
00102 /// <returns><see langword="true"/> if both coordinates of the <see cref="Point"/>s are closer than
      <paramref name="tolerance"/> or if their relative difference (i.e.  <c>(a - b) / (a + b) * 2</c>) is
      smaller than <paramref name="tolerance"/>.  <see langword="false"/> otherwise.</returns>
00103         public bool IsEqual(Point p2, double tolerance)
00104         {
00105             return (Math.Abs(p2.X - this.X) <= tolerance || Math.Abs((p2.X - this.X) / (p2.X +
      this.X)) <= tolerance * 0.5) && (Math.Abs(p2.Y - this.Y) <= tolerance || Math.Abs((p2.Y - this.Y) /
      (p2.Y + this.Y)) <= tolerance * 0.5);
00106         }
00107 
00108 /// <summary>
00109 /// Computes the top-left corner of the <see cref="Rectangle"/> identified by two <see
      cref="Point"/>s.
00110 /// </summary>
00111 /// <param name="p1">The first point.</param>
00112 /// <param name="p2">The second point.</param>
```

```
00113 /// <returns>A <see cref="Point"/> whose <see cref="X"/> coordinate is the smallest between the one of
      <paramref name="p1"/> and <paramref name="p2"/>, and likewise for the <see cref="Y"/>
      coordinate.</returns>
00114         public static Point Min(Point p1, Point p2)
00115         {
00116             return new Point(Math.Min(p1.X, p2.X), Math.Min(p1.Y, p2.Y));
00117         }
00118 
00119 /// <summary>
00120 /// Computes the bottom-right corner of the <see cref="Rectangle"/> identified by two <see
      cref="Point"/>s.
00121 /// </summary>
00122 /// <param name="p1">The first point.</param>
00123 /// <param name="p2">The second point.</param>
00124 /// <returns>A <see cref="Point"/> whose <see cref="X"/> coordinate is the largest between the one of
      <paramref name="p1"/> and <paramref name="p2"/>, and likewise for the <see cref="Y"/>
      coordinate.</returns>
00125         public static Point Max(Point p1, Point p2)
00126         {
00127             return new Point(Math.Max(p1.X, p2.X), Math.Max(p1.Y, p2.Y));
00128         }
00129 
00130 /// <summary>
00131 /// Computes the smallest <see cref="Rectangle"/> that contains all the specified points.
00132 /// </summary>
00133 /// <param name="points">The points whose bounds are being computed.</param>
00134 /// <returns>The smallest <see cref="Rectangle"/> that contains all the specified points.</returns>
00135         public static Rectangle Bounds(IEnumerable<Point> points)
00136         {
00137             bool initialised = false;
00138             Point min = new Point(double.NaN, double.NaN);
00139             Point max = new Point(double.NaN, double.NaN);
00140 
00141             foreach (Point pt in points)
00142             {
00143                 if (!initialised)
00144                 {
00145                     min = pt;
00146                     max = pt;
00147                     initialised = true;
00148                 }
00149                 else
00150                 {
00151                     min = Point.Min(min, pt);
00152                     max = Point.Max(max, pt);
00153                 }
00154             }
00155 
00156             return new Rectangle(min, max);
00157         }
00158 
00159 /// <summary>
00160 /// Computes the smallest <see cref="Rectangle"/> that contains all the specified points.
00161 /// </summary>
00162 /// <param name="points">The points whose bounds are being computed.</param>
00163 /// <returns>The smallest <see cref="Rectangle"/> that contains all the specified points.</returns>
00164         public static Rectangle Bounds(params Point[] points)
00165         {
00166             return Bounds((IEnumerable<Point>)points);
00167         }
00168 
00169 /// <inheritdoc/>
00170         public IEnumerator<double> GetEnumerator()
00171         {
00172             yield return this.X;
00173             yield return this.Y;
00174         }
00175 
00176 /// <inheritdoc/>
00177         IEnumerator IEnumerable.GetEnumerator() => GetEnumerator();
00178 
00179 /// <summary>
00180 /// Implicit conversion to a tuple.
00181 /// </summary>
00182 /// <param name="point">The point to convert.</param>
00183         public static implicit operator (double X, double Y)(Point point)
00184         {
00185             return (point.X, point.Y);
00186         }
00187 
00188 /// <summary>
00189 /// Implicit conversion to an array.
00190 /// </summary>
00191 /// <param name="point">The point to convert</param>
00192         public static implicit operator double[](Point point)
00193         {
00194             return new double[] { point.X, point.Y };
```

```
00195         }
00196
00197 /// <summary>
00198 /// Implicit conversion from a tuple.
00199 /// </summary>
00200 /// <param name="tuple">The tuple to convert.</param>
00201         public static implicit operator Point((double X, double Y) tuple)
00202         {
00203             return new Point(tuple.X, tuple.Y);
00204         }
00205
00206 /// <summary>
00207 /// Explicit conversion from an array (will throw an exception if the array does not contain exactly
      two elements).
00208 /// </summary>
00209 /// <param name="array">The array to convert.  It must contain exactly two elements.</param>
00210         public static explicit operator Point(double[] array)
00211         {
00212             if (array == null)
00213             {
00214                 throw new ArgumentNullException("array", "The array to convert into a Point must not
      be null!");
00215             }
00216             else if (array.Length != 2)
00217             {
00218                 throw new ArgumentOutOfRangeException("array", array.Length.ToString(), "The length of
      the array to convert into a Point must be exactly 2!");
00219             }
00220
00221             return new Point(array[0], array[1]);
00222         }
00223
00224 /// <summary>
00225 /// Subtract the coordinates of two points.
00226 /// </summary>
00227 /// <param name="p1">The first point.</param>
00228 /// <param name="p2">The second point.</param>
00229 /// <returns>A new <see cref="Point"/> whose coordinates are the difference between the coordinates of
      the original points.</returns>
00230         public static Point operator -(Point p1, Point p2)
00231         {
00232             return new Point(p1.X - p2.X, p1.Y - p2.Y);
00233         }
00234
00235 /// <summary>
00236 /// Add the coordinates of two points.
00237 /// </summary>
00238 /// <param name="p1">The first point.</param>
00239 /// <param name="p2">The second point.</param>
00240 /// <returns>A new <see cref="Point"/> whose coordinates are the sum of the coordinates of the
      original points.</returns>
00241         public static Point operator +(Point p1, Point p2)
00242         {
00243             return new Point(p1.X + p2.X, p1.Y + p2.Y);
00244         }
00245
00246 /// <summary>
00247 /// Multiply the coordinates of a point by a scalar.
00248 /// </summary>
00249 /// <param name="t">The scalar.</param>
00250 /// <param name="p">The point.</param>
00251 /// <returns>A new <see cref="Point"/> whose coordinates are the product of the original points' and
      the scalar.</returns>
00252         public static Point operator *(double t, Point p)
00253         {
00254             return new Point(t * p.X, t * p.Y);
00255         }
00256
00257 /// <summary>
00258 /// Multiply the coordinates of a point by a scalar.
00259 /// </summary>
00260 /// <param name="t">The scalar.</param>
00261 /// <param name="p">The point.</param>
00262 /// <returns>A new <see cref="Point"/> whose coordinates are the product of the original points' and
      the scalar.</returns>
00263         public static Point operator *(Point p, double t)
00264         {
00265             return new Point(t * p.X, t * p.Y);
00266         }
00267     }
00268
00269 /// <summary>
00270 /// Represents the size of an object.
00271 /// </summary>
00272     [DebuggerDisplay("\\{ Width:  {Width}, Height:  {Height} \\}")]
00273     public struct Size
00274     {
```

```
00275 /// <summary>
00276 /// Width of the object.
00277 /// </summary>
00278         public double Width;
00279 
00280 /// <summary>
00281 /// Height of the object.
00282 /// </summary>
00283         public double Height;
00284 
00285 /// <summary>
00286 /// Create a new <see cref="Size"/>.
00287 /// </summary>
00288 /// <param name="width">The width of the object.</param>
00289 /// <param name="height">The height of the object.</param>
00290         public Size(double width, double height)
00291         {
00292             Width = width;
00293             Height = height;
00294         }
00295     }
00296 
00297 /// <summary>
00298 /// Represents a rectangle.
00299 /// </summary>
00300     [DebuggerDisplay("\\{ Location:  {Location}, Size:  {Size} \\}")]
00301     public struct Rectangle
00302     {
00303 /// <summary>
00304 /// A rectangle whose dimensions are all <see cref="double.NaN"/>.
00305 /// </summary>
00306         public static readonly Rectangle NaN = new Rectangle(double.NaN, double.NaN, double.NaN,
      double.NaN);
00307 
00308 /// <summary>
00309 /// Checks whether the rectangle is equivalent to <see cref="Rectangle.NaN"/>.
00310 /// </summary>
00311 /// <returns><see langword="true"/> if all the dimensions of the rectangle are <see
      cref="double.NaN"/>, <see langword="false"/> otherwise.</returns>
00312         public bool IsNaN()
00313         {
00314             return double.IsNaN(this.Location.X) && double.IsNaN(this.Location.Y) &&
      double.IsNaN(this.Size.Width) && double.IsNaN(this.Size.Height);
00315         }
00316 
00317 /// <summary>
00318 /// The top-left corner of the rectangle.
00319 /// </summary>
00320         public Point Location;
00321 
00322 /// <summary>
00323 /// The size of the rectangle.
00324 /// </summary>
00325         public Size Size;
00326 
00327 /// <summary>
00328 /// The centre of the rectangle.
00329 /// </summary>
00330         public Point Centre { get { return new Point(this.Location.X + this.Size.Width * 0.5,
      this.Location.Y + this.Size.Height * 0.5); } }
00331 
00332 /// <summary>
00333 /// Create a new <see cref="Rectangle"/> given its top-left corner and its size.
00334 /// </summary>
00335 /// <param name="location">The top-left corner of the rectangle.</param>
00336 /// <param name="size">The size of the rectangle.</param>
00337         public Rectangle(Point location, Size size)
00338         {
00339             this.Location = location;
00340             this.Size = size;
00341         }
00342 
00343 /// <summary>
00344 /// Create a new <see cref="Rectangle"/> given its top-left corner and its size.
00345 /// </summary>
00346 /// <param name="x">The horizontal coordinate of the top-left corner of the rectangle.</param>
00347 /// <param name="y">The vertical coordinate of the top-left corner of the rectangle.</param>
00348 /// <param name="width">The width of the rectangle.</param>
00349 /// <param name="height">The height of the rectangle.</param>
00350         public Rectangle(double x, double y, double width, double height)
00351         {
00352             Location = new Point(x, y);
00353             Size = new Size(width, height);
00354         }
00355 
00356 /// <summary>
00357 /// Create a new <see cref="Rectangle"/> given its top-left corner and its bottom-right corner.
```

```
00358 /// </summary>
00359 /// <param name="topLeft">The top-left corner of the rectangle.</param>
00360 /// <param name="bottomRight">The bottom-right corner of the rectangle.</param>
00361         public Rectangle(Point topLeft, Point bottomRight)
00362         {
00363             this.Location = topLeft;
00364             this.Size = new Size(bottomRight.X - topLeft.X, bottomRight.Y - topLeft.Y);
00365         }
00366 
00367 /// <summary>
00368 /// Computes the rectangular bounds of the union of two <see cref="Rectangle"/>s.
00369 /// </summary>
00370 /// <param name="rectangle1">The first <see cref="Rectangle"/>.</param>
00371 /// <param name="rectangle2">The second <see cref="Rectangle"/>.</param>
00372 /// <returns>The smallest <see cref="Rectangle"/> containing both <paramref name="rectangle1"/> and
      <paramref name="rectangle2"/>.</returns>
00373         public static Rectangle Union(Rectangle rectangle1, Rectangle rectangle2)
00374         {
00375             double minX = rectangle1.Location.X;
00376             double minY = rectangle1.Location.Y;
00377             double maxX = rectangle1.Location.X;
00378             double maxY = rectangle1.Location.Y;
00379 
00380             minX = Math.Min(minX, rectangle1.Location.X + rectangle1.Size.Width);
00381             minX = Math.Min(minX, rectangle2.Location.X);
00382             minX = Math.Min(minX, rectangle2.Location.X + rectangle2.Size.Width);
00383 
00384             minY = Math.Min(minY, rectangle1.Location.Y + rectangle1.Size.Height);
00385             minY = Math.Min(minY, rectangle2.Location.Y);
00386             minY = Math.Min(minY, rectangle2.Location.Y + rectangle2.Size.Height);
00387 
00388 
00389             maxX = Math.Max(maxX, rectangle1.Location.X + rectangle1.Size.Width);
00390             maxX = Math.Max(maxX, rectangle2.Location.X);
00391             maxX = Math.Max(maxX, rectangle2.Location.X + rectangle2.Size.Width);
00392 
00393             maxY = Math.Max(maxY, rectangle1.Location.Y + rectangle1.Size.Height);
00394             maxY = Math.Max(maxY, rectangle2.Location.Y);
00395             maxY = Math.Max(maxY, rectangle2.Location.Y + rectangle2.Size.Height);
00396 
00397             return new Rectangle(minX, minY, maxX - minX, maxY - minY);
00398         }
00399 
00400 /// <summary>
00401 /// Computes the intersection of two <see cref="Rectangle"/>s.
00402 /// </summary>
00403 /// <param name="rectangle1">The first <see cref="Rectangle"/>.</param>
00404 /// <param name="rectangle2">The second <see cref="Rectangle"/>.</param>
00405 /// <returns>The rectangle corresponding to the intersection of <paramref name="rectangle1"/> and
      <paramref name="rectangle2"/>, or <see cref="Rectangle.NaN"/> if the intersection is empty.</returns>
00406         public static Rectangle Intersection(Rectangle rectangle1, Rectangle rectangle2)
00407         {
00408             double x0 = Math.Max(rectangle1.Location.X, rectangle2.Location.X);
00409             double x1 = Math.Min(rectangle1.Location.X + rectangle1.Size.Width, rectangle2.Location.X
      + rectangle2.Size.Width);
00410 
00411             double y0 = Math.Max(rectangle1.Location.Y, rectangle2.Location.Y);
00412             double y1 = Math.Min(rectangle1.Location.Y + rectangle1.Size.Height, rectangle2.Location.Y
      + rectangle2.Size.Height);
00413 
00414             if (x1 >= x0 && y1 >= y0)
00415             {
00416                 return new Rectangle(x0, y0, x1 - x0, y1 - y0);
00417             }
00418             else
00419             {
00420                 return Rectangle.NaN;
00421             }
00422 
00423         }
00424 
00425 /// <summary>
00426 /// Computes the rectangular bounds of the union of multiple <see cref="Rectangle"/>s.
00427 /// </summary>
00428 /// <param name="rectangles">The <see cref="Rectangle"/>s whose union will be computed.</param>
00429 /// <returns>The smallest <see cref="Rectangle"/> containing all the <paramref
      name="rectangles"/>.</returns>
00430         public static Rectangle Union(IEnumerable<Rectangle> rectangles)
00431         {
00432             if (rectangles.Any())
00433             {
00434                 bool initialised = false;
00435 
00436                 Rectangle tbr = new Rectangle();
00437 
00438                 foreach (Rectangle rect in rectangles)
00439                 {
```

```
00440                     if (!initialised)
00441                     {
00442                         tbr = rect;
00443                     }
00444                     else
00445                     {
00446                         tbr = Union(rect, tbr);
00447                     }
00448                 }
00449 
00450                 return tbr;
00451             }
00452             else
00453             {
00454                 return Rectangle.NaN;
00455             }
00456         }
00457 
00458 /// <summary>
00459 /// Computes the rectangular bounds of the union of multiple <see cref="Rectangle"/>s.
00460 /// </summary>
00461 /// <param name="rectangles">The <see cref="Rectangle"/>s whose union will be computed.</param>
00462 /// <returns>The smallest <see cref="Rectangle"/> containing all the <paramref
    name="rectangles"/>.</returns>
00463         public static Rectangle Union(params Rectangle[] rectangles)
00464         {
00465             return Union((IEnumerable<Rectangle>)rectangles);
00466         }
00467     }
00468 }
```

8.89 RasterImage.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System;
00019 using System.Diagnostics;
00020 using System.IO;
00021 using System.Runtime.InteropServices;
00022 
00023 namespace VectSharp
00024 {
00025 /// <summary>
00026 /// Represents the pixel format of a raster image.
00027 /// </summary>
00028     public enum PixelFormats
00029     {
00030 /// <summary>
00031 /// RGB 24bpp format.
00032 /// </summary>
00033         RGB,
00034 
00035 /// <summary>
00036 /// RGBA 32bpp format.
00037 /// </summary>
00038         RGBA,
00039 
00040 /// <summary>
00041 /// BGR 24bpp format.
00042 /// </summary>
00043         BGR,
00044 
00045 /// <summary>
00046 /// BGR 32bpp format.
00047 /// </summary>
00048         BGRA
00049     }
00050 
```

```
00051 /// <summary>
00052 /// An <see cref="IDisposable"/> wrapper around an <see cref="IntPtr"/> that frees the allocated
      memory when it is disposed.
00053 /// </summary>
00054     public class DisposableIntPtr :  IDisposable
00055     {
00056 /// <summary>
00057 /// The pointer to the unmanaged memory.
00058 /// </summary>
00059         public readonly IntPtr InternalPointer;
00060 
00061 /// <summary>
00062 /// Create a new DisposableIntPtr.
00063 /// </summary>
00064 /// <param name="pointer">The pointer that should be freed upon disposing of this object.</param>
00065         public DisposableIntPtr(IntPtr pointer)
00066         {
00067             this.InternalPointer = pointer;
00068         }
00069 
00070         private bool disposedValue;
00071 
00072 ///<inheritdoc/>
00073         protected virtual void Dispose(bool disposing)
00074         {
00075             if (!disposedValue)
00076             {
00077                 Marshal.FreeHGlobal(InternalPointer);
00078                 disposedValue = true;
00079             }
00080         }
00081 
00082 ///<inheritdoc/>
00083         ~DisposableIntPtr()
00084         {
00085             Dispose(disposing:  false);
00086         }
00087 
00088 ///<inheritdoc/>
00089         public void Dispose()
00090         {
00091             Dispose(disposing:  true);
00092             GC.SuppressFinalize(this);
00093         }
00094     }
00095 
00096 /// <summary>
00097 /// Represents a raster image, created from raw pixel data.  Consider using the derived classes
      included in the NuGet package "VectSharp.MuPDFUtils" if you need to load a raster image from a file or
      a <see cref="Stream"/>.
00098 /// </summary>
00099     [DebuggerDisplay("\\{ Width:  {Width}, Height:  {Height} \\}")]
00100     public class RasterImage :  IDisposable
00101     {
00102 /// <summary>
00103 /// The memory address of the image pixel data.
00104 /// </summary>
00105         public IntPtr ImageDataAddress { get; protected set; }
00106 
00107 /// <summary>
00108 /// An <see cref="IDisposable"/> that will be disposed when the image is disposed.
00109 /// </summary>
00110         public IDisposable DataHolder { get; protected set; }
00111 
00112 /// <summary>
00113 /// A univocal identifier for this image.
00114 /// </summary>
00115         public string Id { get; protected set; }
00116 
00117 /// <summary>
00118 /// Determines whether the image has an alpha channel.
00119 /// </summary>
00120         public bool HasAlpha { get; protected set; }
00121 
00122 /// <summary>
00123 /// The width in pixels of the image.
00124 /// </summary>
00125         public int Width { get; protected set; }
00126 
00127 /// <summary>
00128 /// The height in pixels of the image.
00129 /// </summary>
00130         public int Height { get; protected set; }
00131 
00132 /// <summary>
00133 /// Determines whether the image should be interpolated when it is resized.
00134 /// </summary>
```

```
00135         public bool Interpolate { get; protected set; }
00136
00137         private MemoryStream _PNGStream = null;
00138
00139 /// <summary>
00140 /// Contains a representation of the image in PNG format.  Generated at the first access and cached
      until the image is disposed.
00141 /// </summary>
00142         public MemoryStream PNGStream
00143         {
00144             get
00145             {
00146                 if (_PNGStream == null)
00147                 {
00148                     _PNGStream = new MemoryStream();
00149                     this.EncodeAsPNG(_PNGStream);
00150                 }
00151                 _PNGStream.Seek(0, SeekOrigin.Begin);
00152                 return _PNGStream;
00153             }
00154         }
00155
00156 /// <summary>
00157 /// Default constructor, necessary for inheritance.
00158 /// </summary>
00159         protected RasterImage()
00160         {
00161
00162         }
00163
00164 /// <summary>
00165 /// Creates a new <see cref="RasterImage"/> instance from the specified pixel data in RGB or RGBA
      format.
00166 /// </summary>
00167 /// <param name="pixelData">The address of the image pixel data in RGB or RGBA format.</param>
00168 /// <param name="width">The width in pixels of the image.</param>
00169 /// <param name="height">The height in pixels of the image.</param>
00170 /// <param name="hasAlpha">true if the image is in RGBA format, false if it is in RGB format.</param>
00171 /// <param name="interpolate">Whether the image should be interpolated when it is resized.</param>
00172         public RasterImage(IntPtr pixelData, int width, int height, bool hasAlpha, bool interpolate)
00173         {
00174             this.Id = Guid.NewGuid().ToString();
00175             this.ImageDataAddress = pixelData;
00176             this.Width = width;
00177             this.Height = height;
00178             this.HasAlpha = hasAlpha;
00179             this.Interpolate = interpolate;
00180         }
00181
00182 /// <summary>
00183 /// Creates a new <see cref="RasterImage"/> instance from the specified pixel data in RGB or RGBA
      format.
00184 /// </summary>
00185 /// <param name="pixelData">The address of the image pixel data in RGB or RGBA format wrapped in a
      <see cref="DisposableIntPtr"/>.  The <see cref="RasterImage"/> will take ownership of this
      memory.</param>
00186 /// <param name="width">The width in pixels of the image.</param>
00187 /// <param name="height">The height in pixels of the image.</param>
00188 /// <param name="hasAlpha">true if the image is in RGBA format, false if it is in RGB format.</param>
00189 /// <param name="interpolate">Whether the image should be interpolated when it is resized.</param>
00190         public RasterImage(ref DisposableIntPtr pixelData, int width, int height, bool hasAlpha, bool
      interpolate)
00191         {
00192             this.Id = Guid.NewGuid().ToString();
00193             this.ImageDataAddress = pixelData.InternalPointer;
00194             this.DataHolder = pixelData;
00195             this.Width = width;
00196             this.Height = height;
00197             this.HasAlpha = hasAlpha;
00198             this.Interpolate = interpolate;
00199         }
00200
00201 /// <summary>
00202 /// Creates a new <see cref="RasterImage"/> instance copying the specified pixel data.
00203 /// </summary>
00204 /// <param name="data">The image pixel data that will be copied.</param>
00205 /// <param name="width">The width in pixels of the image.</param>
00206 /// <param name="height">The height in pixels of the image.</param>
00207 /// <param name="pixelFormat">The format of the pixel data.</param>
00208 /// <param name="interpolate">Whether the image should be interpolated when it is resized.</param>
00209         public RasterImage(byte[] data, int width, int height, PixelFormats pixelFormat, bool
      interpolate)
00210         {
00211             this.ImageDataAddress = Marshal.AllocHGlobal(data.Length);
00212             GC.AddMemoryPressure(data.Length);
00213             this.DataHolder = new DisposableIntPtr(this.ImageDataAddress);
00214             this.Id = Guid.NewGuid().ToString();
```

```
00215             this.Width = width;
00216             this.Height = height;
00217             this.Interpolate = interpolate;
00218 
00219             switch (pixelFormat)
00220             {
00221                 case PixelFormats.RGB:
00222                 case PixelFormats.RGBA:
00223                     Marshal.Copy(data, 0, this.ImageDataAddress, data.Length);
00224                     this.HasAlpha = pixelFormat == PixelFormats.RGBA;
00225                     break;
00226 
00227                 case PixelFormats.BGRA:
00228                 case PixelFormats.BGR:
00229                     this.HasAlpha = pixelFormat == PixelFormats.BGRA;
00230 
00231                     int pixelSize = pixelFormat == PixelFormats.BGRA ? 4 :  3;
00232                     int pixelCount = width * height;
00233 
00234                     unsafe
00235                     {
00236                         byte* dataPointer = (byte*)this.ImageDataAddress;
00237                         for (int i = 0; i < pixelCount; i++)
00238                         {
00239                             dataPointer[i * pixelSize] = data[i * pixelSize + 2];
00240                             dataPointer[i * pixelSize + 1] = data[i * pixelSize + 1];
00241                             dataPointer[i * pixelSize + 2] = data[i * pixelSize];
00242                             if (pixelSize == 4)
00243                             {
00244                                 dataPointer[i * pixelSize + 3] = data[i * pixelSize + 3];
00245                             }
00246                         }
00247                     }
00248                     break;
00249             }
00250         }
00251 
00252         private void EncodeAsPNG(Stream outputStream)
00253         {
00254             unsafe
00255             {
00256                 PNGEncoder.SavePNG((byte*)this.ImageDataAddress, this.Width, this.Height,
      this.HasAlpha, outputStream, PNGEncoder.FilterModes.Adaptive);
00257             }
00258         }
00259 
00260 /// <summary>
00261 /// Disposes the <see cref="PNGStream"/>.  Also useful if is is necessary to regenerate it, e.g.
      because the underlying image pixel data has changed.
00262 /// </summary>
00263         public void ClearPNGCache()
00264         {
00265             _PNGStream?.Dispose();
00266             _PNGStream = null;
00267         }
00268 
00269         private bool disposedValue;
00270 
00271 /// <inheritdoc/>
00272         protected virtual void Dispose(bool disposing)
00273         {
00274             if (!disposedValue)
00275             {
00276                 if (disposing)
00277                 {
00278                     _PNGStream?.Dispose();
00279 
00280                     if (DataHolder != null)
00281                     {
00282                         DataHolder.Dispose();
00283                         GC.RemoveMemoryPressure(Height * Width * (HasAlpha ?  4 :  3));
00284                     }
00285                 }
00286                 disposedValue = true;
00287             }
00288         }
00289 
00290 ///<inheritdoc/>
00291         ~RasterImage()
00292         {
00293             Dispose(disposing:  false);
00294         }
00295 
00296 /// <inheritdoc/>
00297         public void Dispose()
00298         {
00299             Dispose(disposing:  true);
```

```
00300             GC.SuppressFinalize(this);
00301         }
00302     }
00303 }
```

8.90 Segment.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Diagnostics.Contracts;
00021 using System.Linq;
00022
00023 namespace VectSharp
00024 {
00025
00026 /// <summary>
00027 /// Represents a segment as part of a <see cref="GraphicsPath"/>.
00028 /// </summary>
00029     public abstract class Segment
00030     {
00031
00032 /// <summary>
00033 /// The type of the <see cref="Segment"/>.
00034 /// </summary>
00035         public abstract SegmentType Type { get; }
00036
00037 /// <summary>
00038 /// The points used to define the <see cref="Segment"/>.
00039 /// </summary>
00040         public Point[] Points { get; protected set; }
00041
00042 /// <summary>
00043 /// The end point of the <see cref="Segment"/>.
00044 /// </summary>
00045         public virtual Point Point
00046         {
00047             get
00048             {
00049                 return Points[Points.Length - 1];
00050             }
00051         }
00052
00053 /// <summary>
00054 /// Creates a copy of the <see cref="Segment"/>.
00055 /// </summary>
00056 /// <returns>A copy of the <see cref="Segment"/>.</returns>
00057         [Pure]
00058         public abstract Segment Clone();
00059
00060 /// <summary>
00061 /// Computes the length of the <see cref="Segment"/>.
00062 /// </summary>
00063 /// <param name="previousPoint">The point from which the <see cref="Segment"/> starts (i.e.  the
      endpoint of the previous <see cref="Segment"/>).</param>
00064 /// <returns>The length of the segment.</returns>
00065         public abstract double Measure(Point previousPoint);
00066
00067 /// <summary>
00068 /// Gets the point on the <see cref="Segment"/> at the specified (relative) <paramref
      name="position"/>).
00069 /// </summary>
00070 /// <param name="previousPoint">The point from which the <see cref="Segment"/> starts (i.e.  the
      endpoint of the previous <see cref="Segment"/>).</param>
00071 /// <param name="position">The relative position on the <see cref="Segment"/> (0 is the start of the
      <see cref="Segment"/>, 1 is the end of the <see cref="Segment"/>).</param>
00072 /// <returns>The point at the specified position.</returns>
```

```
00073         public abstract Point GetPointAt(Point previousPoint, double position);
00074
00075 /// <summary>
00076 /// Gets the tangent to the <see cref="Segment"/> at the specified (relative) <paramref
      name="position"/>).
00077 /// </summary>
00078 /// <param name="previousPoint">The point from which the <see cref="Segment"/> starts (i.e.  the
      endpoint of the previous <see cref="Segment"/>).</param>
00079 /// <param name="position">The relative position on the <see cref="Segment"/> (0 is the start of the
      <see cref="Segment"/>, 1 is the end of the <see cref="Segment"/>).</param>
00080 /// <returns>The tangent to the point at the specified position.</returns>
00081         public abstract Point GetTangentAt(Point previousPoint, double position);
00082
00083 /// <summary>
00084 /// Transform the segment into a series of linear segments.  Segments that are already linear are not
      changed.
00085 /// </summary>
00086 /// <param name="previousPoint">The point from which the <see cref="Segment"/> starts (i.e.  the
      endpoint of the previous <see cref="Segment"/>).</param>
00087 /// <param name="resolution">The absolute length between successive samples in curve segments.</param>
00088 /// <returns>A collection of linear segments that approximate the current segment.</returns>
00089         [Pure]
00090         public abstract IEnumerable<Segment> Linearise(Point?  previousPoint, double resolution);
00091
00092 /// <summary>
00093 /// Gets the tanget at the points at which the segment would be linearised.
00094 /// </summary>
00095 /// <param name="previousPoint">The point from which the <see cref="Segment"/> starts (i.e.  the
      endpoint of the previous <see cref="Segment"/>).</param>
00096 /// <param name="resolution">The absolute length between successive samples in curve segments.</param>
00097 /// <returns>A collection of tangents at the points in which the segment would be
      linearised.</returns>
00098         public abstract IEnumerable<Point> GetLinearisationTangents(Point?  previousPoint, double
      resolution);
00099
00100 /// <summary>
00101 /// Applies an arbitrary transformation to all of the points of the <see cref="Segment"/>.
00102 /// </summary>
00103 /// <param name="transformationFunction">An arbitrary transformation function.</param>
00104 /// <returns>A collection of <see cref="Segment"/>s that have been transformed according to the
      <paramref name="transformationFunction"/>.</returns>
00105         [Pure]
00106         public abstract IEnumerable<Segment> Transform(Func<Point, Point> transformationFunction);
00107
00108 /// <summary>
00109 /// Flattens the <see cref="Segment"/>, replacing curve segments with series of line segments that
      approximate them, ensuring the specified maximum deviation from the original path.
00110 /// </summary>
00111 /// <param name="previousPoint">The point from which the <see cref="Segment"/> starts (i.e.  the
      endpoint of the previous <see cref="Segment"/>).</param>
00112 /// <param name="flatness">The maximum deviation from the original path.</param>
00113 /// <returns>A collection of <see cref="Segment"/>s composed only of linear segments that approximates
      the current <see cref="Segment"/>.</returns>
00114         [Pure]
00115         public abstract IEnumerable<Segment> Flatten(Point?  previousPoint, double flatness);
00116
00117 /// <summary>
00118 /// Flattens the <see cref="Segment"/>, replacing curve segments with series of line segments that
      approximate them, ensuring the specified maximum deviation from the original path, assuming that the
      <see cref="Segment"/> will be drawn with the specified <paramref name="offset"/>.
00119 /// </summary>
00120 /// <param name="previousPoint">The point from which the <see cref="Segment"/> starts (i.e.  the
      endpoint of the previous <see cref="Segment"/>).</param>
00121 /// <param name="offset">The offset that will be used to draw the <see cref="Segment"/> (e.g., the
      line width of the stroke).</param>
00122 /// <param name="flatness">The maximum deviation from the original path.</param>
00123 /// <returns>A collection of tuples where the first element is a <see cref="Point"/> representing the
      end-point of a linear segment, and the second element is a <see cref="Point"/> containing the value of
      the tangent to the original <see cref="Segment"/> at that point.</returns>
00124         public abstract IEnumerable<(Point point, Point tangent)>
      FlattenForOffsetAndGetTangents(Point?  previousPoint, double offset, double flatness);
00125     }
00126
00127     internal class MoveSegment :  Segment
00128     {
00129         public override SegmentType Type => SegmentType.Move;
00130
00131         public MoveSegment(Point p)
00132         {
00133             this.Points = new Point[] { p };
00134         }
00135
00136         public MoveSegment(double x, double y)
00137         {
00138             this.Points = new Point[] { new Point(x, y) };
00139         }
00140
```

```
00141         public override Segment Clone()
00142         {
00143             return new MoveSegment(this.Point);
00144         }
00145 
00146         public override double Measure(Point previousPoint)
00147         {
00148             return 0;
00149         }
00150 
00151         public override Point GetPointAt(Point previousPoint, double position)
00152         {
00153             throw new InvalidOperationException();
00154         }
00155 
00156         public override Point GetTangentAt(Point previousPoint, double position)
00157         {
00158             throw new InvalidOperationException();
00159         }
00160 
00161         public override IEnumerable<Segment> Linearise(Point?  previousPoint, double resolution)
00162         {
00163             yield return new MoveSegment(this.Point);
00164         }
00165 
00166         public override IEnumerable<Point> GetLinearisationTangents(Point?  previousPoint, double
      resolution)
00167         {
00168             throw new InvalidOperationException();
00169         }
00170 
00171         public override IEnumerable<Segment> Transform(Func<Point, Point> transformationFunction)
00172         {
00173             yield return new MoveSegment(transformationFunction(this.Point));
00174         }
00175 
00176         public override IEnumerable<Segment> Flatten(Point?  previousPoint, double flatness)
00177         {
00178             yield return new MoveSegment(this.Point);
00179         }
00180 
00181         public override IEnumerable<(Point point, Point tangent)>
      FlattenForOffsetAndGetTangents(Point?  previousPoint, double offset, double flatness)
00182         {
00183             throw new InvalidOperationException();
00184         }
00185     }
00186 
00187     internal class LineSegment :  Segment
00188     {
00189         public override SegmentType Type => SegmentType.Line;
00190 
00191         public LineSegment(Point p)
00192         {
00193             this.Points = new Point[] { p };
00194         }
00195 
00196         public LineSegment(double x, double y)
00197         {
00198             this.Points = new Point[] { new Point(x, y) };
00199         }
00200 
00201         public override Segment Clone()
00202         {
00203             return new LineSegment(this.Point);
00204         }
00205 
00206         private double cachedLength = double.NaN;
00207 
00208         public override double Measure(Point previousPoint)
00209         {
00210             if (double.IsNaN(cachedLength))
00211             {
00212                 cachedLength = Math.Sqrt((this.Point.X - previousPoint.X) * (this.Point.X -
      previousPoint.X) + (this.Point.Y - previousPoint.Y) * (this.Point.Y - previousPoint.Y));
00213             }
00214 
00215             return cachedLength;
00216         }
00217 
00218         public override Point GetPointAt(Point previousPoint, double position)
00219         {
00220             return new Point(previousPoint.X * (1 - position) + this.Point.X * position,
      previousPoint.Y * (1 - position) + this.Point.Y * position);
00221         }
00222 
00223         public override Point GetTangentAt(Point previousPoint, double position)
```

```
00224         {
00225             return new Point(this.Point.X - previousPoint.X, this.Point.Y -
      previousPoint.Y).Normalize();
00226         }
00227
00228         public override IEnumerable<Segment> Linearise(Point?  previousPoint, double resolution)
00229         {
00230             yield return new LineSegment(this.Point);
00231         }
00232
00233         public override IEnumerable<Point> GetLinearisationTangents(Point?  previousPoint, double
      resolution)
00234         {
00235             yield return this.GetTangentAt(previousPoint.Value, 1);
00236         }
00237
00238         public override IEnumerable<Segment> Transform(Func<Point, Point> transformationFunction)
00239         {
00240             yield return new LineSegment(transformationFunction(this.Point));
00241         }
00242
00243         public override IEnumerable<Segment> Flatten(Point?  previousPoint, double flatness)
00244         {
00245             yield return new LineSegment(this.Point);
00246         }
00247
00248         public override IEnumerable<(Point point, Point tangent)>
      FlattenForOffsetAndGetTangents(Point?  previousPoint, double offset, double flatness)
00249         {
00250             if (previousPoint != null)
00251             {
00252                 yield return (previousPoint.Value, this.GetTangentAt(previousPoint.Value, 1));
00253             }
00254
00255             yield return (this.Point, this.GetTangentAt(previousPoint.Value, 1));
00256         }
00257     }
00258
00259     internal class CloseSegment :  Segment
00260     {
00261         public override SegmentType Type => SegmentType.Close;
00262
00263         public CloseSegment() { }
00264
00265         public override Segment Clone()
00266         {
00267             return new CloseSegment();
00268         }
00269
00270         public override double Measure(Point previousPoint)
00271         {
00272             return 0;
00273         }
00274
00275         public override Point GetPointAt(Point previousPoint, double position)
00276         {
00277             throw new InvalidOperationException();
00278         }
00279
00280         public override Point GetTangentAt(Point previousPoint, double position)
00281         {
00282             throw new InvalidOperationException();
00283         }
00284
00285         public override IEnumerable<Segment> Linearise(Point?  previousPoint, double resolution)
00286         {
00287             yield return new CloseSegment();
00288         }
00289
00290         public override IEnumerable<Point> GetLinearisationTangents(Point?  previousPoint, double
      resolution)
00291         {
00292             throw new InvalidOperationException();
00293         }
00294
00295         public override IEnumerable<Segment> Transform(Func<Point, Point> transformationFunction)
00296         {
00297             yield return new CloseSegment();
00298         }
00299
00300         public override IEnumerable<Segment> Flatten(Point?  previousPoint, double flatness)
00301         {
00302             yield return new CloseSegment();
00303         }
00304
00305         public override IEnumerable<(Point point, Point tangent)>
      FlattenForOffsetAndGetTangents(Point?  previousPoint, double offset, double flatness)
```

```
00306         {
00307             throw new InvalidOperationException();
00308         }
00309     }
00310
00311     internal class CubicBezierSegment :  Segment
00312     {
00313         public override SegmentType Type => SegmentType.CubicBezier;
00314         public CubicBezierSegment(double x1, double y1, double x2, double y2, double x3, double y3)
00315         {
00316             Points = new Point[] { new Point(x1, y1), new Point(x2, y2), new Point(x3, y3) };
00317         }
00318
00319         public CubicBezierSegment(Point p1, Point p2, Point p3)
00320         {
00321             Points = new Point[] { p1, p2, p3 };
00322         }
00323
00324         public override Segment Clone()
00325         {
00326             return new CubicBezierSegment(Points[0], Points[1], Points[2]);
00327         }
00328
00329         private double cachedLength = double.NaN;
00330         private int cachedSegments = -1;
00331
00332         public override double Measure(Point previousPoint)
00333         {
00334             if (double.IsNaN(cachedLength))
00335             {
00336                 int segments = 16;
00337                 double prevLength = 0;
00338                 double currLength = Measure(previousPoint, segments);
00339
00340                 while (currLength > 0.00001 && Math.Abs(currLength - prevLength) / currLength >
    0.0001)
00341                 {
00342                     segments *= 2;
00343                     prevLength = currLength;
00344                     currLength = Measure(previousPoint, segments);
00345                 }
00346
00347                 cachedSegments = segments;
00348
00349                 cachedLength = currLength;
00350             }
00351
00352             return cachedLength;
00353         }
00354
00355         public Point GetBezierPointAt(Point previousPoint, double position)
00356         {
00357             if (position <= 1 && position >= 0)
00358             {
00359                 return new Point(
00360                 this.Points[2].X * position * position * position + 3 * this.Points[1].X * position *
    position * (1 - position) + 3 * this.Points[0].X * position * (1 - position) * (1 - position) +
    previousPoint.X * (1 - position) * (1 - position) * (1 - position),
00361                 this.Points[2].Y * position * position * position + 3 * this.Points[1].Y * position *
    position * (1 - position) + 3 * this.Points[0].Y * position * (1 - position) * (1 - position) +
    previousPoint.Y * (1 - position) * (1 - position) * (1 - position)
00362                 );
00363             }
00364             else if (position > 1)
00365             {
00366                 Point tangent = GetBezierTangentAt(previousPoint, 1);
00367
00368                 double excessLength = (position - 1) * this.Measure(previousPoint);
00369
00370                 return new Point(this.Point.X + tangent.X * excessLength, this.Point.Y + tangent.Y *
    excessLength);
00371             }
00372             else
00373             {
00374                 Point tangent = GetBezierTangentAt(previousPoint, 0);
00375
00376                 return new Point(previousPoint.X + tangent.X * position * this.Measure(previousPoint),
    previousPoint.Y + tangent.Y * position * this.Measure(previousPoint));
00377             }
00378         }
00379
00380         public override Point GetPointAt(Point previousPoint, double position)
00381         {
00382             double t = GetTFromPosition(previousPoint, position);
00383             return this.GetBezierPointAt(previousPoint, t);
00384         }
00385
```

```
00386         public override Point GetTangentAt(Point previousPoint, double position)
00387         {
00388             double t = GetTFromPosition(previousPoint, position);
00389             return this.GetBezierTangentAt(previousPoint, t);
00390         }
00391 
00392         private double Measure(Point startPoint, int segments)
00393         {
00394             double delta = 1.0 / segments;
00395 
00396             double tbr = 0;
00397 
00398             for (int i = 1; i < segments; i++)
00399             {
00400                 Point p1 = GetBezierPointAt(startPoint, delta * (i - 1));
00401                 Point p2 = GetBezierPointAt(startPoint, delta * i);
00402 
00403                 tbr += Math.Sqrt((p1.X - p2.X) * (p1.X - p2.X) + (p1.Y - p2.Y) * (p1.Y - p2.Y));
00404             }
00405 
00406             return tbr;
00407         }
00408 
00409         private double Measure(Point startPoint, int segments, double maxT)
00410         {
00411             double delta = maxT / segments;
00412 
00413             double tbr = 0;
00414 
00415             for (int i = 1; i < segments; i++)
00416             {
00417                 Point p1 = GetBezierPointAt(startPoint, delta * (i - 1));
00418                 Point p2 = GetBezierPointAt(startPoint, delta * i);
00419 
00420                 tbr += Math.Sqrt((p1.X - p2.X) * (p1.X - p2.X) + (p1.Y - p2.Y) * (p1.Y - p2.Y));
00421             }
00422 
00423             return tbr;
00424         }
00425 
00426         private Point GetBezierTangentAt(Point previousPoint, double position)
00427         {
00428             if (position <= 1 && position >= 0)
00429             {
00430                 if (position == 0 && previousPoint.IsEqual(this.Points[0], GraphicsPath.Tolerance))
00431                 {
00432                     return (this.Points[1] - previousPoint).Normalize();
00433                 }
00434                 else if (position == 1 && this.Points[2].IsEqual(this.Points[1],
      GraphicsPath.Tolerance))
00435                 {
00436                     return (this.Points[2] - this.Points[0]).Normalize();
00437                 }
00438                 else
00439                 {
00440                     return new Point(
00441                         3 * this.Points[2].X * position * position +
00442                         3 * this.Points[1].X * position * (2 - 3 * position) +
00443                         3 * this.Points[0].X * (3 * position * position - 4 * position + 1) +
00444                         -3 * previousPoint.X * (1 - position) * (1 - position),
00445 
00446                         3 * this.Points[2].Y * position * position +
00447                         3 * this.Points[1].Y * position * (2 - 3 * position) +
00448                         3 * this.Points[0].Y * (3 * position * position - 4 * position + 1) +
00449                         -3 * previousPoint.Y * (1 - position) * (1 - position)).Normalize();
00450                 }
00451             }
00452             else if (position > 1)
00453             {
00454                 return GetBezierTangentAt(previousPoint, 1);
00455             }
00456             else
00457             {
00458                 return GetBezierTangentAt(previousPoint, 0);
00459             }
00460         }
00461 
00462         private double GetTFromPosition(Point previousPoint, double position)
00463         {
00464             if (position <= 0 || position >= 1)
00465             {
00466                 return position;
00467             }
00468             else
00469             {
00470                 double length = this.Measure(previousPoint);
00471 
```

```
00472                 double lowerBound = 0;
00473                 double upperBound = 0.5;
00474 
00475                 double lowerPos = 0;
00476                 double upperPos = Measure(previousPoint, (int)Math.Ceiling(this.cachedSegments *
      upperBound), upperBound) / length;
00477 
00478                 if (upperPos < position)
00479                 {
00480                     lowerBound = upperBound;
00481                     lowerPos = upperPos;
00482 
00483                     upperBound = 1;
00484                     upperPos = 1;
00485                 }
00486 
00487                 while (Math.Min(upperPos - position, position - lowerPos) > 0.001)
00488                 {
00489                     double mid = (lowerBound + upperBound) * 0.5;
00490                     double midPos = Measure(previousPoint, (int)Math.Ceiling(this.cachedSegments *
      mid), mid) / length;
00491 
00492                     if (midPos > position)
00493                     {
00494                         upperBound = mid;
00495                         upperPos = midPos;
00496                     }
00497                     else
00498                     {
00499                         lowerBound = mid;
00500                         lowerPos = midPos;
00501                     }
00502                 }
00503 
00504                 return lowerBound + (position - lowerPos) / (upperPos - lowerPos) * (upperBound -
      lowerBound);
00505             }
00506         }
00507 
00508         public override IEnumerable<Segment> Linearise(Point?  previousPoint, double resolution)
00509         {
00510             double length = this.Measure(previousPoint.Value);
00511             int segmentCount = (int)Math.Ceiling(length / resolution);
00512 
00513             for (int i = 0; i < segmentCount; i++)
00514             {
00515                 yield return new LineSegment(this.GetPointAt(previousPoint.Value, (double)(i + 1) /
      segmentCount));
00516             }
00517         }
00518 
00519         public override IEnumerable<Point> GetLinearisationTangents(Point?  previousPoint, double
      resolution)
00520         {
00521             double length = this.Measure(previousPoint.Value);
00522             int segmentCount = (int)Math.Ceiling(length / resolution);
00523 
00524             for (int i = 0; i < segmentCount; i++)
00525             {
00526                 yield return this.GetTangentAt(previousPoint.Value, (double)(i + 1) / segmentCount);
00527             }
00528         }
00529 
00530         public override IEnumerable<Segment> Transform(Func<Point, Point> transformationFunction)
00531         {
00532             yield return new CubicBezierSegment(transformationFunction(this.Points[0]),
      transformationFunction(this.Points[1]), transformationFunction(this.Points[2]));
00533         }
00534 
00535         private (CubicBezierSegment first, CubicBezierSegment second) Subdivide(Point previousPoint,
      double t)
00536         {
00537             Point p0 = previousPoint + t * (this.Points[0] - previousPoint);
00538             Point p1 = this.Points[0] + t * (this.Points[1] - this.Points[0]);
00539             Point p2 = this.Points[1] + t * (this.Points[2] - this.Points[1]);
00540             Point p0_2 = p0 + t * (p1 - p0);
00541             Point p1_2 = p1 + t * (p2 - p1);
00542             Point p0_3 = p0_2 + t * (p1_2 - p0_2);
00543 
00544             return (new CubicBezierSegment(p0, p0_2, p0_3), new CubicBezierSegment(p1_2, p2,
      this.Points[2]));
00545         }
00546 
00547         // Based on https://doi.org/10.1016/j.cag.2005.08.002
00548         private IEnumerable<Segment> FlattenPrivate(Point?  previousPoint, double flatness)
00549         {
00550             Point startingPoint;
```

```
00551
00552             startingPoint = previousPoint ??  this.Points[0];
00553
00554             double s2 = ((this.Points[1].X - startingPoint.X) * (this.Points[0].Y - startingPoint.Y) -
      (this.Points[1].Y - startingPoint.Y) * (this.Points[0].X - startingPoint.X)) /
      Math.Sqrt((this.Points[0].X - startingPoint.X) * (this.Points[0].X - startingPoint.X) +
      (this.Points[0].Y - startingPoint.Y) * (this.Points[0].Y - startingPoint.Y));
00555
00556             double t = 2 * Math.Sqrt(flatness / (3 * Math.Abs(s2)));
00557
00558             if (t < 1)
00559             {
00560                 (CubicBezierSegment first, CubicBezierSegment second) = this.Subdivide(startingPoint,
      t);
00561
00562                 yield return new LineSegment(first.Points[2]);
00563
00564                 foreach (Segment seg in second.FlattenPrivate(first.Points[2], flatness))
00565                 {
00566                     yield return seg;
00567                 }
00568             }
00569             else
00570             {
00571                 yield return new LineSegment(this.Points[2]);
00572             }
00573         }
00574
00575         // Based on https://doi.org/10.1016/j.cag.2005.08.002
00576         public override IEnumerable<(Point point, Point tangent)>
      FlattenForOffsetAndGetTangents(Point?  previousPoint, double offset, double flatness)
00577         {
00578             return FlattenForOffsetAndGetTangents(previousPoint, offset, flatness, false);
00579         }
00580
00581         // Based on https://doi.org/10.1016/j.cag.2005.08.002
00582         private IEnumerable<(Point point, Point tangent)> FlattenForOffsetAndGetTangents(Point?
      previousPoint, double offset, double flatness, bool skipFirst)
00583         {
00584             Point startingPoint;
00585
00586             startingPoint = previousPoint ??  this.Points[0];
00587
00588             if (!skipFirst)
00589             {
00590                 yield return (startingPoint, this.GetBezierTangentAt(startingPoint, 0));
00591             }
00592
00593             double r1sq = (this.Points[0].X - startingPoint.X) * (this.Points[0].X - startingPoint.X)
      + (this.Points[0].Y - startingPoint.Y) * (this.Points[0].Y - startingPoint.Y);
00594             double s2 = ((this.Points[1].X - startingPoint.X) * (this.Points[0].Y - startingPoint.Y) -
      (this.Points[1].Y - startingPoint.Y) * (this.Points[0].X - startingPoint.X)) /
      Math.Sqrt((this.Points[0].X - startingPoint.X) * (this.Points[0].X - startingPoint.X) +
      (this.Points[0].Y - startingPoint.Y) * (this.Points[0].Y - startingPoint.Y));
00595
00596             if (double.IsNaN(s2))
00597             {
00598                 s2 = ((this.Points[2].X - startingPoint.X) * (this.Points[1].Y - startingPoint.Y) -
      (this.Points[2].Y - startingPoint.Y) * (this.Points[1].X - startingPoint.X)) /
      Math.Sqrt((this.Points[1].X - startingPoint.X) * (this.Points[1].X - startingPoint.X) +
      (this.Points[1].Y - startingPoint.Y) * (this.Points[1].Y - startingPoint.Y));
00599             }
00600
00601             if (double.IsNaN(s2))
00602             {
00603                 s2 = 1;
00604             }
00605
00606             double t = 2 * Math.Sqrt(Math.Abs(flatness / (3 * Math.Abs(s2) * (1 - offset * s2 / (3 *
      r1sq)))));
00607
00608             if (r1sq < 1e-7)
00609             {
00610                 t = 2 * Math.Sqrt(Math.Abs(flatness / (3 * Math.Abs(s2))));
00611
00612                 if (this.Points[1].X == startingPoint.X && this.Points[1].Y == startingPoint.Y &&
      this.Points[2].X == startingPoint.X && this.Points[2].Y == startingPoint.Y)
00613                 {
00614                     yield return (this.Point, new Point(double.NaN, double.NaN));
00615                     yield break;
00616                 }
00617             }
00618
00619             if (t < 1)
00620             {
00621                 (CubicBezierSegment first, CubicBezierSegment second) = this.Subdivide(startingPoint,
      t);
```

```
00622
00623                 yield return (first.Points[2], this.GetBezierTangentAt(startingPoint, t));
00624
00625                 foreach ((Point, Point) p in second.FlattenForOffsetAndGetTangents(first.Points[2],
      offset, flatness, true))
00626                 {
00627                     yield return p;
00628                 }
00629             }
00630             else
00631             {
00632                 yield return (this.Points[2], this.GetBezierTangentAt(startingPoint, 1));
00633             }
00634         }
00635
00636         internal IEnumerable<CubicBezierSegment> MonotoniseOnY(Point previousPoint)
00637         {
00638             double a = previousPoint.Y;
00639             double b = this.Points[0].Y;
00640             double c = this.Points[1].Y;
00641             double d = this.Points[2].Y;
00642
00643             if ((a + 3 * c != 3 * b + d))
00644             {
00645                 double t1 = ((-6 * a + 12 * b - 6 * c) + Math.Sqrt((6 * a - 12 * b + 6 * c) * (6 * a -
      12 * b + 6 * c) - 4 * (3 * b - 3 * a) * (-3 * a + 9 * b - 9 * c + 3 * d))) / (2 * (-3 * a + 9 * b - 9
      * c + 3 * d));
00646                 double t2 = ((-6 * a + 12 * b - 6 * c) - Math.Sqrt((6 * a - 12 * b + 6 * c) * (6 * a -
      12 * b + 6 * c) - 4 * (3 * b - 3 * a) * (-3 * a + 9 * b - 9 * c + 3 * d))) / (2 * (-3 * a + 9 * b - 9
      * c + 3 * d));
00647
00648                 if (t1 > 0 && t1 < 1 && t2 > 0 && t2 < 1)
00649                 {
00650                     (CubicBezierSegment seg1, CubicBezierSegment seg2) = this.Subdivide(previousPoint,
      Math.Min(t1, t2));
00651
00652                     yield return seg1;
00653
00654                     foreach (CubicBezierSegment seg in seg2.MonotoniseOnY(seg1.Point))
00655                     {
00656                         yield return seg;
00657                     }
00658                 }
00659                 else if (t1 > 0 && t1 < 1)
00660                 {
00661                     (CubicBezierSegment seg1, CubicBezierSegment seg2) = this.Subdivide(previousPoint,
      t1);
00662
00663                     yield return seg1;
00664                     yield return seg2;
00665                 }
00666                 else if (t2 > 0 && t2 < 1)
00667                 {
00668                     (CubicBezierSegment seg1, CubicBezierSegment seg2) = this.Subdivide(previousPoint,
      t2);
00669
00670                     yield return seg1;
00671                     yield return seg2;
00672                 }
00673                 else
00674                 {
00675                     yield return this;
00676                 }
00677             }
00678             else
00679             {
00680                 double t = (a - b) / (2 * (a - 2 * b + c));
00681
00682                 if (t > 0 && t < 1)
00683                 {
00684                     (CubicBezierSegment seg1, CubicBezierSegment seg2) = this.Subdivide(previousPoint,
      t);
00685
00686                     yield return seg1;
00687                     yield return seg2;
00688                 }
00689                 else
00690                 {
00691                     yield return this;
00692                 }
00693             }
00694         }
00695
00696         // Based on https://doi.org/10.1016/j.cag.2005.08.002
00697         public override IEnumerable<Segment> Flatten(Point?  previousPoint, double flatness)
00698         {
00699             Point currPoint = previousPoint ??  this.Points[0];
```

```
00700
00701             foreach (Segment seg in this.BreakAtInflectionPoints(previousPoint, flatness))
00702             {
00703                 if (seg is CubicBezierSegment cub)
00704                 {
00705                     foreach (Segment seg2 in cub.FlattenPrivate(currPoint, flatness))
00706                     {
00707                         yield return seg2;
00708                     }
00709                 }
00710                 else
00711                 {
00712                     yield return seg;
00713                 }
00714                 currPoint = seg.Point;
00715             }
00716         }
00717
00718         // Based on https://doi.org/10.1016/j.cag.2005.08.002
00719         internal IEnumerable<Segment> BreakAtInflectionPoints(Point?  previousPoint, double flatness)
00720         {
00721             Point startingPoint;
00722
00723             startingPoint = previousPoint ??  this.Points[0];
00724
00725             double ax = -startingPoint.X + 3 * this.Points[0].X - 3 * this.Points[1].X +
      this.Points[2].X;
00726             double ay = -startingPoint.Y + 3 * this.Points[0].Y - 3 * this.Points[1].Y +
      this.Points[2].Y;
00727             double bx = 3 * startingPoint.X - 6 * this.Points[0].X + 3 * this.Points[1].X;
00728             double by = 3 * startingPoint.Y - 6 * this.Points[0].Y + 3 * this.Points[1].Y;
00729             double cx = -3 * startingPoint.X + 3 * this.Points[0].X;
00730             double cy = -3 * startingPoint.Y + 3 * this.Points[0].Y;
00731
00732             double tcusp = -0.5 * (ay * cx - ax * cy) / (ay * bx - ax * by);
00733
00734             double t1 = tcusp - Math.Sqrt(tcusp * tcusp - (by * cx - bx * cy) / (ay * bx - ax * by) /
      3);
00735             double t2 = tcusp + Math.Sqrt(tcusp * tcusp - (by * cx - bx * cy) / (ay * bx - ax * by) /
      3);
00736
00737             List<double> validInflectionPoints = new List<double>();
00738
00739             if (t1 >= 0 && t1 <= 1)
00740             {
00741                 validInflectionPoints.Add(t1);
00742             }
00743
00744             if (t2 >= 0 && t2 <= 1 && (validInflectionPoints.Count == 0 || t2 -
      validInflectionPoints[0] > 1e-5))
00745             {
00746                 validInflectionPoints.Add(t2);
00747             }
00748
00749             if (validInflectionPoints.Count == 0)
00750             {
00751                 yield return this;
00752             }
00753             else if (validInflectionPoints.Count == 1)
00754             {
00755                 (CubicBezierSegment first, CubicBezierSegment second) = this.Subdivide(startingPoint,
      validInflectionPoints[0]);
00756
00757                 double s3 = Math.Abs(((second.Points[2].X - first.Point.X) * (second.Points[0].Y -
      first.Point.Y) - (second.Points[2].Y - first.Point.Y) * (second.Points[0].X - first.Point.X)) /
      Math.Sqrt((second.Points[0].X - first.Point.X) * (second.Points[0].X - first.Point.X) +
      (second.Points[0].Y - first.Point.Y) * (second.Points[0].Y - first.Point.Y)));
00758
00759                 double tf = Math.Pow(flatness / s3, 1.0 / 3);
00760
00761                 if (double.IsNaN(tf) || double.IsInfinity(tf))
00762                 {
00763                     tf = 0;
00764                 }
00765
00766                 double tm = Math.Max(0, validInflectionPoints[0] - tf * (1 -
      validInflectionPoints[0]));
00767                 double tp = Math.Min(validInflectionPoints[0] + tf * (1 - validInflectionPoints[0]),
      1);
00768
00769                 if (tm > 0 && tp < 1)
00770                 {
00771                     (first, _) = this.Subdivide(startingPoint, tm);
00772
00773                     CubicBezierSegment temp;
00774                     (temp, second) = this.Subdivide(startingPoint, tp);
00775
```

```
00776                     yield return first;
00777
00778                     if (tm != tp)
00779                     {
00780                         yield return new LineSegment(temp.Point);
00781                     }
00782
00783                     yield return second;
00784                 }
00785                 else if (tm == 0 && tp < 1)
00786                 {
00787                     (first, second) = this.Subdivide(startingPoint, tp);
00788
00789                     yield return new LineSegment(first.Point);
00790
00791                     yield return second;
00792                 }
00793                 else if (tm > 0 && tp == 1)
00794                 {
00795                     (first, _) = this.Subdivide(startingPoint, tm);
00796
00797                     yield return first;
00798
00799                     if (tm != tp)
00800                     {
00801                         yield return new LineSegment(this.Point);
00802                     }
00803                 }
00804                 else
00805                 {
00806                     yield return new LineSegment(this.Point);
00807                 }
00808             }
00809             else if (validInflectionPoints.Count == 2)
00810             {
00811                 (CubicBezierSegment first, CubicBezierSegment second) = this.Subdivide(startingPoint,
      validInflectionPoints[0]);
00812
00813                 double s3 = Math.Abs(((second.Points[2].X - first.Point.X) * (second.Points[0].Y -
      first.Point.Y) - (second.Points[2].Y - first.Point.Y) * (second.Points[0].X - first.Point.X)) /
      Math.Sqrt((second.Points[0].X - first.Point.X) * (second.Points[0].X - first.Point.X) +
      (second.Points[0].Y - first.Point.Y) * (second.Points[0].Y - first.Point.Y)));
00814
00815                 double tf = Math.Pow(flatness / s3, 1.0 / 3);
00816
00817                 if (double.IsNaN(tf) || double.IsInfinity(tf))
00818                 {
00819                     tf = 0;
00820                 }
00821
00822                 double t1m = Math.Max(validInflectionPoints[0] - tf * (1 - validInflectionPoints[0]),
      0);
00823                 double t1p = Math.Min(validInflectionPoints[0] + tf * (1 - validInflectionPoints[0]),
      validInflectionPoints[1]);
00824
00825
00826                 (first, second) = this.Subdivide(startingPoint, validInflectionPoints[1]);
00827
00828                 s3 = Math.Abs(((second.Points[2].X - first.Point.X) * (second.Points[0].Y -
      first.Point.Y) - (second.Points[2].Y - first.Point.Y) * (second.Points[0].X - first.Point.X)) /
      Math.Sqrt((second.Points[0].X - first.Point.X) * (second.Points[0].X - first.Point.X) +
      (second.Points[0].Y - first.Point.Y) * (second.Points[0].Y - first.Point.Y)));
00829
00830                 tf = Math.Pow(flatness / s3, 1.0 / 3);
00831
00832                 if (double.IsNaN(tf) || double.IsInfinity(tf))
00833                 {
00834                     tf = 0;
00835                 }
00836
00837                 double t2m = Math.Max(validInflectionPoints[1] - tf * (1 - validInflectionPoints[1]),
      t1p);
00838                 double t2p = Math.Min(validInflectionPoints[1] + tf * (1 - validInflectionPoints[1]),
      1);
00839
00840                 if (t1m > 0)
00841                 {
00842                     (first, _) = this.Subdivide(startingPoint, t1m);
00843                     yield return first;
00844                 }
00845
00846                 if (t1p > t1m)
00847                 {
00848                     CubicBezierSegment temp;
00849                     (temp, _) = this.Subdivide(startingPoint, t1p);
00850                     yield return new LineSegment(temp.Point);
00851                 }
```

```
00852
00853                 if (t2m > t1p)
00854                 {
00855                     CubicBezierSegment temp, third;
00856                     (temp, second) = this.Subdivide(startingPoint, t1p);
00857                     (third, _) = second.Subdivide(temp.Point, (t2m - t1p) / (1 - t1p));
00858                     yield return third;
00859                 }
00860
00861                 if (t2p > t2m)
00862                 {
00863                     CubicBezierSegment temp;
00864                     (temp, second) = this.Subdivide(startingPoint, t1p);
00865                     (temp, _) = second.Subdivide(temp.Point, (t2p - t1p) / (1 - t1p));
00866                     yield return new LineSegment(temp.Point);
00867                 }
00868
00869                 if (t2p < 1)
00870                 {
00871                     CubicBezierSegment temp, fourth;
00872                     (temp, second) = this.Subdivide(startingPoint, t1p);
00873                     (_, fourth) = second.Subdivide(temp.Point, (t2p - t1p) / (1 - t1p));
00874                     yield return fourth;
00875                 }
00876             }
00877         }
00878     }
00879
00880     internal class ArcSegment :  Segment
00881     {
00882         public override SegmentType Type => SegmentType.Arc;
00883
00884         public Segment[] ToBezierSegments()
00885         {
00886             List<Segment> tbr = new List<Segment>();
00887
00888             if (EndAngle > StartAngle)
00889             {
00890                 if (EndAngle - StartAngle <= Math.PI / 2)
00891                 {
00892                     tbr.AddRange(GetBezierSegment(Points[0].X, Points[0].Y, Radius, StartAngle,
      EndAngle, true));
00893                 }
00894                 else
00895                 {
00896                     int count = (int)Math.Ceiling(2 * (EndAngle - StartAngle) / Math.PI);
00897                     double angle = StartAngle;
00898
00899                     for (int i = 0; i < count; i++)
00900                     {
00901                         tbr.AddRange(GetBezierSegment(Points[0].X, Points[0].Y, Radius, angle, angle +
      (EndAngle - StartAngle) / count, i == 0));
00902                         angle += (EndAngle - StartAngle) / count;
00903                     }
00904                 }
00905             }
00906             else if (EndAngle < StartAngle)
00907             {
00908                 Point startPoint = new Point(Points[0].X + Radius * Math.Cos(EndAngle), Points[0].Y +
      Radius * Math.Sin(EndAngle));
00909                 if (StartAngle - EndAngle <= Math.PI / 2)
00910                 {
00911                     tbr.AddRange(GetBezierSegment(Points[0].X, Points[0].Y, Radius, EndAngle,
      StartAngle, true));
00912                 }
00913                 else
00914                 {
00915                     int count = (int)Math.Ceiling(2 * (StartAngle - EndAngle) / Math.PI);
00916                     double angle = EndAngle;
00917
00918                     for (int i = 0; i < count; i++)
00919                     {
00920                         tbr.AddRange(GetBezierSegment(Points[0].X, Points[0].Y, Radius, angle, angle +
      (StartAngle - EndAngle) / count, i == 0));
00921                         angle += (StartAngle - EndAngle) / count;
00922                     }
00923                 }
00924
00925                 return ReverseSegments(tbr, startPoint).ToArray();
00926             }
00927
00928             return tbr.ToArray();
00929         }
00930
00931         private static Segment[] ReverseSegments(IReadOnlyList<Segment> originalSegments, Point
      startPoint)
00932         {
```

```
00933             List<Segment> tbr = new List<Segment>(originalSegments.Count);
00934
00935             for (int i = originalSegments.Count - 1; i >= 0; i--)
00936             {
00937                 switch (originalSegments[i].Type)
00938                 {
00939                     case SegmentType.Line:
00940                         if (i > 0)
00941                         {
00942                             tbr.Add(new LineSegment(originalSegments[i - 1].Point));
00943                         }
00944                         else
00945                         {
00946                             tbr.Add(new LineSegment(startPoint));
00947                         }
00948                         break;
00949                     case SegmentType.CubicBezier:
00950                         CubicBezierSegment originalSegment = (CubicBezierSegment)originalSegments[i];
00951                         if (i > 0)
00952                         {
00953                             tbr.Add(new CubicBezierSegment(originalSegment.Points[1],
      originalSegment.Points[0], originalSegments[i - 1].Point));
00954                         }
00955                         else
00956                         {
00957                             tbr.Add(new CubicBezierSegment(originalSegment.Points[1],
      originalSegment.Points[0], startPoint));
00958                         }
00959                         break;
00960                 }
00961             }
00962
00963             return tbr.ToArray();
00964         }
00965
00966         const double k = 0.55191496;
00967
00968         private static Segment[] GetBezierSegment(double cX, double cY, double radius, double
      startAngle, double endAngle, bool firstArc)
00969         {
00970             double phi = Math.PI / 4;
00971
00972             double x1 = radius * Math.Cos(phi);
00973             double y1 = radius * Math.Sin(phi);
00974
00975             double x4 = x1;
00976             double y4 = -y1;
00977
00978             double x3 = x1 + k * radius * Math.Sin(phi);
00979             double y3 = y1 - k * radius * Math.Cos(phi);
00980
00981             double x2 = x4 + k * radius * Math.Sin(phi);
00982             double y2 = y4 + k * radius * Math.Cos(phi);
00983
00984             double u = 2 * (endAngle - startAngle) / Math.PI;
00985
00986             double fx2 = (1 - u) * x4 + u * x2;
00987             double fy2 = (1 - u) * y4 + u * y2;
00988
00989             double fx3 = (1 - u) * fx2 + u * ((1 - u) * x2 + u * x3);
00990             double fy3 = (1 - u) * fy2 + u * ((1 - u) * y2 + u * y3);
00991
00992             double rX1 = cX + radius * Math.Cos(startAngle);
00993             double rY1 = cY + radius * Math.Sin(startAngle);
00994
00995             double rX4 = cX + radius * Math.Cos(endAngle);
00996             double rY4 = cY + radius * Math.Sin(endAngle);
00997
00998             Point rot2 = Utils.RotatePoint(new Point(fx2, fy2), phi + startAngle);
00999             Point rot3 = Utils.RotatePoint(new Point(fx3, fy3), phi + startAngle);
01000
01001             List<Segment> tbr = new List<Segment>();
01002
01003             if (firstArc)
01004             {
01005                 tbr.Add(new LineSegment(rX1, rY1));
01006             }
01007
01008             tbr.Add(new CubicBezierSegment(cX + rot2.X, cY + rot2.Y, cX + rot3.X, cY + rot3.Y, rX4,
      rY4));
01009
01010             return tbr.ToArray();
01011         }
01012         public double Radius { get; }
01013         public double StartAngle { get; }
01014         public double EndAngle { get; }
01015
```

```
01016         public ArcSegment(Point center, double radius, double startAngle, double endAngle)
01017         {
01018             this.Points = new Point[] { center };
01019             this.Radius = radius;
01020             this.StartAngle = startAngle;
01021             this.EndAngle = endAngle;
01022         }
01023 
01024         public ArcSegment(double centerX, double centerY, double radius, double startAngle, double
      endAngle)
01025         {
01026             this.Points = new Point[] { new Point(centerX, centerY) };
01027             this.Radius = radius;
01028             this.StartAngle = startAngle;
01029             this.EndAngle = endAngle;
01030         }
01031 
01032         public override Segment Clone()
01033         {
01034             return new ArcSegment(Point.X, Point.Y, Radius, StartAngle, EndAngle);
01035         }
01036 
01037         public override Point Point
01038         {
01039             get
01040             {
01041                 return new Point(this.Points[0].X + Math.Cos(EndAngle) * Radius, this.Points[0].Y +
      Math.Sin(EndAngle) * Radius);
01042             }
01043         }
01044 
01045 
01046         private double cachedLength = double.NaN;
01047 
01048         public override double Measure(Point previousPoint)
01049         {
01050             if (double.IsNaN(cachedLength))
01051             {
01052                 Point arcStartPoint = new Point(this.Points[0].X + Math.Cos(StartAngle) * Radius,
      this.Points[0].Y + Math.Sin(StartAngle) * Radius);
01053 
01054                 cachedLength = Radius * Math.Abs(EndAngle - StartAngle) + Math.Sqrt((arcStartPoint.X -
      previousPoint.X) * (arcStartPoint.X - previousPoint.X) + (arcStartPoint.Y - previousPoint.Y) *
      (arcStartPoint.Y - previousPoint.Y));
01055             }
01056 
01057             return cachedLength;
01058         }
01059 
01060         public override Point GetPointAt(Point previousPoint, double position)
01061         {
01062             double totalLength = this.Measure(previousPoint);
01063             double arcLength = Radius * Math.Abs(EndAngle - StartAngle);
01064 
01065             double preArc = (totalLength - arcLength) / totalLength;
01066 
01067             if (position < preArc)
01068             {
01069                 if (position >= 0)
01070                 {
01071                     double relPos = position / preArc;
01072                     Point arcStartPoint = new Point(this.Points[0].X + Math.Cos(StartAngle) * Radius,
      this.Points[0].Y + Math.Sin(StartAngle) * Radius);
01073 
01074                     return new Point(previousPoint.X * (1 - relPos) + arcStartPoint.X * relPos,
      previousPoint.Y * (1 - relPos) + arcStartPoint.Y * relPos);
01075                 }
01076                 else
01077                 {
01078                     Point arcStartPoint = new Point(this.Points[0].X + Math.Cos(StartAngle) * Radius,
      this.Points[0].Y + Math.Sin(StartAngle) * Radius);
01079                     Point tangent = GetTangentAt(previousPoint, 0);
01080                     double excessLength = position * this.Measure(previousPoint);
01081                     return new Point(arcStartPoint.X + tangent.X * excessLength, arcStartPoint.Y +
      tangent.Y * excessLength);
01082                 }
01083             }
01084             else
01085             {
01086                 double relPos = position - preArc / (1 - preArc);
01087 
01088                 if (relPos <= 1)
01089                 {
01090                     double angle = StartAngle * (1 - relPos) + EndAngle * relPos;
01091                     return new Point(this.Points[0].X + Radius * Math.Cos(angle), this.Points[0].Y +
      Radius * Math.Sin(angle));
01092                 }
```

```
01093                 else
01094                 {
01095                     Point arcEndPoint = this.Point;
01096                     Point tangent = GetTangentAt(previousPoint, 1);
01097                     double excessLength = (position - 1) * this.Measure(previousPoint);
01098                     return new Point(arcEndPoint.X + tangent.X * excessLength, arcEndPoint.Y +
      tangent.Y * excessLength);
01099                 }
01100             }
01101         }
01102 
01103         public override Point GetTangentAt(Point previousPoint, double position)
01104         {
01105             double totalLength = this.Measure(previousPoint);
01106             double arcLength = Radius * Math.Abs(EndAngle - StartAngle);
01107 
01108             double preArc = (totalLength - arcLength) / totalLength;
01109 
01110             if (position < preArc)
01111             {
01112                 Point arcStartPoint = new Point(this.Points[0].X + Math.Cos(StartAngle) * Radius,
      this.Points[0].Y + Math.Sin(StartAngle) * Radius);
01113                 Point tang = new Point((arcStartPoint.X - previousPoint.X) * Math.Sign(EndAngle -
      StartAngle), (arcStartPoint.Y - previousPoint.Y) * Math.Sign(EndAngle - StartAngle)).Normalize();
01114 
01115                 if (tang.Modulus() > 0.001)
01116                 {
01117                     return tang.Normalize();
01118                 }
01119                 else
01120                 {
01121                     return this.GetTangentAt(previousPoint, 0);
01122                 }
01123             }
01124             else
01125             {
01126                 double relPos = position - preArc / (1 - preArc);
01127 
01128                 if (relPos <= 1)
01129                 {
01130                     double angle = StartAngle * (1 - relPos) + EndAngle * relPos;
01131                     return new Point(-Math.Sin(angle) * Math.Sign(EndAngle - StartAngle),
      Math.Cos(angle) * Math.Sign(EndAngle - StartAngle));
01132                 }
01133                 else
01134                 {
01135                     return new Point(-Math.Sin(EndAngle) * Math.Sign(EndAngle - StartAngle),
      Math.Cos(EndAngle) * Math.Sign(EndAngle - StartAngle));
01136                 }
01137             }
01138 
01139         }
01140 
01141         public override IEnumerable<Segment> Linearise(Point?  previousPoint, double resolution)
01142         {
01143             double length = this.Measure(previousPoint.Value);
01144             int segmentCount = (int)Math.Ceiling(length / resolution);
01145 
01146             for (int i = 0; i < segmentCount; i++)
01147             {
01148                 yield return new LineSegment(this.GetPointAt(previousPoint.Value, (double)(i + 1) /
      segmentCount));
01149             }
01150         }
01151         public override IEnumerable<Point> GetLinearisationTangents(Point?  previousPoint, double
      resolution)
01152         {
01153             double length = this.Measure(previousPoint.Value);
01154             int segmentCount = (int)Math.Ceiling(length / resolution);
01155 
01156             for (int i = 0; i < segmentCount; i++)
01157             {
01158                 yield return this.GetTangentAt(previousPoint.Value, (double)(i + 1) / segmentCount);
01159             }
01160         }
01161 
01162         public override IEnumerable<Segment> Transform(Func<Point, Point> transformationFunction)
01163         {
01164             foreach (Segment seg in this.ToBezierSegments())
01165             {
01166                 foreach (Segment seg2 in seg.Transform(transformationFunction))
01167                 {
01168                     yield return seg2;
01169                 }
01170             }
01171         }
01172 
```

```
01173         public override IEnumerable<Segment> Flatten(Point?  previousPoint, double flatness)
01174         {
01175             double length = this.Measure(previousPoint.Value);
01176             double resolution = 2 * Math.Sqrt(flatness * (2 * this.Radius - flatness));
01177 
01178             int segmentCount = (int)Math.Ceiling(length / resolution);
01179 
01180             for (int i = 0; i < segmentCount; i++)
01181             {
01182                 yield return new LineSegment(this.GetPointAt(previousPoint.Value, (double)(i + 1) /
      segmentCount));
01183             }
01184         }
01185 
01186         public override IEnumerable<(Point point, Point tangent)>
      FlattenForOffsetAndGetTangents(Point?  previousPoint, double offset, double flatness)
01187         {
01188             double length = this.Measure(previousPoint.Value);
01189             double resolution = 2 * Math.Sqrt(flatness * (2 * (this.Radius + offset) - flatness));
01190 
01191             int segmentCount = (int)Math.Ceiling(length / resolution);
01192 
01193             yield return (this.GetPointAt(previousPoint.Value, 0),
      this.GetTangentAt(previousPoint.Value, 0));
01194 
01195             for (int i = 0; i < segmentCount; i++)
01196             {
01197                 yield return (this.GetPointAt(previousPoint.Value, (double)(i + 1) / segmentCount),
      this.GetTangentAt(previousPoint.Value, (double)(i + 1) / segmentCount));
01198             }
01199         }
01200     }
01201 
01202 
01203 }
```

8.91 SmoothSpline.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017 
00018 using System.Collections.Generic;
00019 using System.Linq;
00020 
00021 namespace VectSharp
00022 {
00023     //derived from https://www.particleincell.com/wp-content/uploads/2012/06/bezier-spline.js
00024 
00025     internal static class SmoothSpline
00026     {
00027         public static Point[] SmoothSplines(Point[] points)
00028         {
00029             double[] x = (from el in points select el.X).ToArray();
00030             double[] y = (from el in points select el.Y).ToArray();
00031 
00032             (double[] p1, double[] p2) px = ComputeControlPoints(x);
00033             (double[] p1, double[] p2) py = ComputeControlPoints(y);
00034 
00035             List<Point> tbr = new List<Point>();
00036 
00037             for (int i = 0; i < points.Length - 1; i++)
00038             {
00039                 tbr.Add(new Point(x[i], y[i]));
00040                 tbr.Add(new Point(px.p1[i], py.p1[i]));
00041                 tbr.Add(new Point(px.p2[i], py.p2[i]));
00042             }
00043 
00044             tbr.Add(new Point(x[x.Length - 1], y[x.Length - 1]));
00045 
```

```
00046             return tbr.ToArray();
00047         }
00048 
00049         /*computes control points given knots K, this is the brain of the operation*/
00050         private static (double[] p1, double[] p2) ComputeControlPoints(double[] K)
00051         {
00052             int n = K.Length - 1;
00053 
00054 
00055             double[] p1 = new double[n];
00056             double[] p2 = new double[n];
00057 
00058 
00059             /*rhs vector*/
00060             double[] a = new double[n];
00061             double[] b = new double[n];
00062             double[] c = new double[n];
00063             double[] r = new double[n];
00064 
00065             /*left most segment*/
00066             a[0] = 0;
00067             b[0] = 2;
00068             c[0] = 1;
00069             r[0] = K[0] + 2 * K[1];
00070 
00071             /*internal segments*/
00072             for (int i = 1; i < n - 1; i++)
00073             {
00074                 a[i] = 1;
00075                 b[i] = 4;
00076                 c[i] = 1;
00077                 r[i] = 4 * K[i] + 2 * K[i + 1];
00078             }
00079 
00080             /*right segment*/
00081             a[n - 1] = 2;
00082             b[n - 1] = 7;
00083             c[n - 1] = 0;
00084             r[n - 1] = 8 * K[n - 1] + K[n];
00085 
00086             /*solves Ax=b with the Thomas algorithm (from Wikipedia)*/
00087             for (int i = 1; i < n; i++)
00088             {
00089                 double m = a[i] / b[i - 1];
00090                 b[i] = b[i] - m * c[i - 1];
00091                 r[i] = r[i] - m * r[i - 1];
00092             }
00093 
00094             p1[n - 1] = r[n - 1] / b[n - 1];
00095             for (int i = n - 2; i >= 0; i--)
00096             {
00097                 p1[i] = (r[i] - c[i] * p1[i + 1]) / b[i];
00098             }
00099 
00100             /*we have p1, now compute p2*/
00101             for (int i = 0; i < n - 1; i++)
00102             {
00103                 p2[i] = 2 * K[i + 1] - p1[i + 1];
00104             }
00105 
00106             p2[n - 1] = 0.5 * (K[n] + p1[n - 1]);
00107 
00108             return (p1, p2);
00109         }
00110 
00111     }
00112 }
```

8.92 StandardColours.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004 
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008 
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU Lesser General Public License for more details.
00013 
```

```
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Text;
00021
00022 namespace VectSharp
00023 {
00024     public partial struct Colour
00025     {
00026         private static Dictionary<string, Colour> StandardColours = new Dictionary<string,
      Colour>(StringComparer.OrdinalIgnoreCase)
00027         {
00028             { "Black", Colour.FromRgb(0, 0, 0) },
00029             { "Navy", Colour.FromRgb(0, 0, 128) },
00030             { "DarkBlue", Colour.FromRgb(0, 0, 139) },
00031             { "MediumBlue", Colour.FromRgb(0, 0, 205) },
00032             { "Blue", Colour.FromRgb(0, 0, 255) },
00033             { "DarkGreen", Colour.FromRgb(0, 100, 0) },
00034             { "Green", Colour.FromRgb(0, 128, 0) },
00035             { "Teal", Colour.FromRgb(0, 128, 128) },
00036             { "DarkCyan", Colour.FromRgb(0, 139, 139) },
00037             { "DeepSkyBlue", Colour.FromRgb(0, 191, 255) },
00038             { "DarkTurquoise", Colour.FromRgb(0, 206, 209) },
00039             { "MediumSpringGreen", Colour.FromRgb(0, 250, 154) },
00040             { "Lime", Colour.FromRgb(0, 255, 0) },
00041             { "SpringGreen", Colour.FromRgb(0, 255, 127) },
00042             { "Aqua", Colour.FromRgb(0, 255, 255) },
00043             { "Cyan", Colour.FromRgb(0, 255, 255) },
00044             { "MidnightBlue", Colour.FromRgb(25, 25, 112) },
00045             { "DodgerBlue", Colour.FromRgb(30, 144, 255) },
00046             { "LightSeaGreen", Colour.FromRgb(32, 178, 170) },
00047             { "ForestGreen", Colour.FromRgb(34, 139, 34) },
00048             { "SeaGreen", Colour.FromRgb(46, 139, 87) },
00049             { "DarkSlateGray", Colour.FromRgb(47, 79, 79) },
00050             { "DarkSlateGrey", Colour.FromRgb(47, 79, 79) },
00051             { "LimeGreen", Colour.FromRgb(50, 205, 50) },
00052             { "MediumSeaGreen", Colour.FromRgb(60, 179, 113) },
00053             { "Turquoise", Colour.FromRgb(64, 224, 208) },
00054             { "RoyalBlue", Colour.FromRgb(65, 105, 225) },
00055             { "SteelBlue", Colour.FromRgb(70, 130, 180) },
00056             { "DarkSlateBlue", Colour.FromRgb(72, 61, 139) },
00057             { "MediumTurquoise", Colour.FromRgb(72, 209, 204) },
00058             { "Indigo", Colour.FromRgb(75, 0, 130) },
00059             { "DarkOliveGreen", Colour.FromRgb(85, 107, 47) },
00060             { "CadetBlue", Colour.FromRgb(95, 158, 160) },
00061             { "CornflowerBlue", Colour.FromRgb(100, 149, 237) },
00062             { "RebeccaPurple", Colour.FromRgb(102, 51, 153) },
00063             { "MediumAquaMarine", Colour.FromRgb(102, 205, 170) },
00064             { "DimGray", Colour.FromRgb(105, 105, 105) },
00065             { "DimGrey", Colour.FromRgb(105, 105, 105) },
00066             { "SlateBlue", Colour.FromRgb(106, 90, 205) },
00067             { "OliveDrab", Colour.FromRgb(107, 142, 35) },
00068             { "SlateGray", Colour.FromRgb(112, 128, 144) },
00069             { "SlateGrey", Colour.FromRgb(112, 128, 144) },
00070             { "LightSlateGray", Colour.FromRgb(119, 136, 153) },
00071             { "LightSlateGrey", Colour.FromRgb(119, 136, 153) },
00072             { "MediumSlateBlue", Colour.FromRgb(123, 104, 238) },
00073             { "LawnGreen", Colour.FromRgb(124, 252, 0) },
00074             { "Chartreuse", Colour.FromRgb(127, 255, 0) },
00075             { "Aquamarine", Colour.FromRgb(127, 255, 212) },
00076             { "Maroon", Colour.FromRgb(128, 0, 0) },
00077             { "Purple", Colour.FromRgb(128, 0, 128) },
00078             { "Olive", Colour.FromRgb(128, 128, 0) },
00079             { "Gray", Colour.FromRgb(128, 128, 128) },
00080             { "Grey", Colour.FromRgb(128, 128, 128) },
00081             { "SkyBlue", Colour.FromRgb(135, 206, 235) },
00082             { "LightSkyBlue", Colour.FromRgb(135, 206, 250) },
00083             { "BlueViolet", Colour.FromRgb(138, 43, 226) },
00084             { "DarkRed", Colour.FromRgb(139, 0, 0) },
00085             { "DarkMagenta", Colour.FromRgb(139, 0, 139) },
00086             { "SaddleBrown", Colour.FromRgb(139, 69, 19) },
00087             { "DarkSeaGreen", Colour.FromRgb(143, 188, 143) },
00088             { "LightGreen", Colour.FromRgb(144, 238, 144) },
00089             { "MediumPurple", Colour.FromRgb(147, 112, 219) },
00090             { "DarkViolet", Colour.FromRgb(148, 0, 211) },
00091             { "PaleGreen", Colour.FromRgb(152, 251, 152) },
00092             { "DarkOrchid", Colour.FromRgb(153, 50, 204) },
00093             { "YellowGreen", Colour.FromRgb(154, 205, 50) },
00094             { "Sienna", Colour.FromRgb(160, 82, 45) },
00095             { "Brown", Colour.FromRgb(165, 42, 42) },
00096             { "DarkGray", Colour.FromRgb(169, 169, 169) },
00097             { "DarkGrey", Colour.FromRgb(169, 169, 169) },
00098             { "LightBlue", Colour.FromRgb(173, 216, 230) },
00099             { "GreenYellow", Colour.FromRgb(173, 255, 47) },
```

```
00100             { "PaleTurquoise", Colour.FromRgb(175, 238, 238) },
00101             { "LightSteelBlue", Colour.FromRgb(176, 196, 222) },
00102             { "PowderBlue", Colour.FromRgb(176, 224, 230) },
00103             { "FireBrick", Colour.FromRgb(178, 34, 34) },
00104             { "DarkGoldenRod", Colour.FromRgb(184, 134, 11) },
00105             { "MediumOrchid", Colour.FromRgb(186, 85, 211) },
00106             { "RosyBrown", Colour.FromRgb(188, 143, 143) },
00107             { "DarkKhaki", Colour.FromRgb(189, 183, 107) },
00108             { "Silver", Colour.FromRgb(192, 192, 192) },
00109             { "MediumVioletRed", Colour.FromRgb(199, 21, 133) },
00110             { "IndianRed", Colour.FromRgb(205, 92, 92) },
00111             { "Peru", Colour.FromRgb(205, 133, 63) },
00112             { "Chocolate", Colour.FromRgb(210, 105, 30) },
00113             { "Tan", Colour.FromRgb(210, 180, 140) },
00114             { "LightGray", Colour.FromRgb(211, 211, 211) },
00115             { "LightGrey", Colour.FromRgb(211, 211, 211) },
00116             { "Thistle", Colour.FromRgb(216, 191, 216) },
00117             { "Orchid", Colour.FromRgb(218, 112, 214) },
00118             { "GoldenRod", Colour.FromRgb(218, 165, 32) },
00119             { "PaleVioletRed", Colour.FromRgb(219, 112, 147) },
00120             { "Crimson", Colour.FromRgb(220, 20, 60) },
00121             { "Gainsboro", Colour.FromRgb(220, 220, 220) },
00122             { "Plum", Colour.FromRgb(221, 160, 221) },
00123             { "BurlyWood", Colour.FromRgb(222, 184, 135) },
00124             { "LightCyan", Colour.FromRgb(224, 255, 255) },
00125             { "Lavender", Colour.FromRgb(230, 230, 250) },
00126             { "DarkSalmon", Colour.FromRgb(233, 150, 122) },
00127             { "Violet", Colour.FromRgb(238, 130, 238) },
00128             { "PaleGoldenRod", Colour.FromRgb(238, 232, 170) },
00129             { "LightCoral", Colour.FromRgb(240, 128, 128) },
00130             { "Khaki", Colour.FromRgb(240, 230, 140) },
00131             { "AliceBlue", Colour.FromRgb(240, 248, 255) },
00132             { "HoneyDew", Colour.FromRgb(240, 255, 240) },
00133             { "Azure", Colour.FromRgb(240, 255, 255) },
00134             { "SandyBrown", Colour.FromRgb(244, 164, 96) },
00135             { "Wheat", Colour.FromRgb(245, 222, 179) },
00136             { "Beige", Colour.FromRgb(245, 245, 220) },
00137             { "WhiteSmoke", Colour.FromRgb(245, 245, 245) },
00138             { "MintCream", Colour.FromRgb(245, 255, 250) },
00139             { "GhostWhite", Colour.FromRgb(248, 248, 255) },
00140             { "Salmon", Colour.FromRgb(250, 128, 114) },
00141             { "AntiqueWhite", Colour.FromRgb(250, 235, 215) },
00142             { "Linen", Colour.FromRgb(250, 240, 230) },
00143             { "LightGoldenRodYellow", Colour.FromRgb(250, 250, 210) },
00144             { "OldLace", Colour.FromRgb(253, 245, 230) },
00145             { "Red", Colour.FromRgb(255, 0, 0) },
00146             { "Fuchsia", Colour.FromRgb(255, 0, 255) },
00147             { "Magenta", Colour.FromRgb(255, 0, 255) },
00148             { "DeepPink", Colour.FromRgb(255, 20, 147) },
00149             { "OrangeRed", Colour.FromRgb(255, 69, 0) },
00150             { "Tomato", Colour.FromRgb(255, 99, 71) },
00151             { "HotPink", Colour.FromRgb(255, 105, 180) },
00152             { "Coral", Colour.FromRgb(255, 127, 80) },
00153             { "DarkOrange", Colour.FromRgb(255, 140, 0) },
00154             { "LightSalmon", Colour.FromRgb(255, 160, 122) },
00155             { "Orange", Colour.FromRgb(255, 165, 0) },
00156             { "LightPink", Colour.FromRgb(255, 182, 193) },
00157             { "Pink", Colour.FromRgb(255, 192, 203) },
00158             { "Gold", Colour.FromRgb(255, 215, 0) },
00159             { "PeachPuff", Colour.FromRgb(255, 218, 185) },
00160             { "NavajoWhite", Colour.FromRgb(255, 222, 173) },
00161             { "Moccasin", Colour.FromRgb(255, 228, 181) },
00162             { "Bisque", Colour.FromRgb(255, 228, 196) },
00163             { "MistyRose", Colour.FromRgb(255, 228, 225) },
00164             { "BlanchedAlmond", Colour.FromRgb(255, 235, 205) },
00165             { "PapayaWhip", Colour.FromRgb(255, 239, 213) },
00166             { "LavenderBlush", Colour.FromRgb(255, 240, 245) },
00167             { "SeaShell", Colour.FromRgb(255, 245, 238) },
00168             { "Cornsilk", Colour.FromRgb(255, 248, 220) },
00169             { "LemonChiffon", Colour.FromRgb(255, 250, 205) },
00170             { "FloralWhite", Colour.FromRgb(255, 250, 240) },
00171             { "Snow", Colour.FromRgb(255, 250, 250) },
00172             { "Yellow", Colour.FromRgb(255, 255, 0) },
00173             { "LightYellow", Colour.FromRgb(255, 255, 224) },
00174             { "Ivory", Colour.FromRgb(255, 255, 240) },
00175             { "White", Colour.FromRgb(255, 255, 255) },
00176         };
00177     }
00178
00179 /// <summary>
00180 /// Standard colours.
00181 /// </summary>
00182     public static class Colours
00183     {
00184 /// <summary>
00185 /// Black #000000
00186 /// </summary>
```

```
00187         public static Colour Black = Colour.FromRgb(0, 0, 0);
00188 /// <summary>
00189 /// Navy #000080
00190 /// </summary>
00191         public static Colour Navy = Colour.FromRgb(0, 0, 128);
00192 /// <summary>
00193 /// DarkBlue #00008B
00194 /// </summary>
00195         public static Colour DarkBlue = Colour.FromRgb(0, 0, 139);
00196 /// <summary>
00197 /// MediumBlue #0000CD
00198 /// </summary>
00199         public static Colour MediumBlue = Colour.FromRgb(0, 0, 205);
00200 /// <summary>
00201 /// Blue #0000FF
00202 /// </summary>
00203         public static Colour Blue = Colour.FromRgb(0, 0, 255);
00204 /// <summary>
00205 /// DarkGreen #006400
00206 /// </summary>
00207         public static Colour DarkGreen = Colour.FromRgb(0, 100, 0);
00208 /// <summary>
00209 /// Green #008000
00210 /// </summary>
00211         public static Colour Green = Colour.FromRgb(0, 128, 0);
00212 /// <summary>
00213 /// Teal #008080
00214 /// </summary>
00215         public static Colour Teal = Colour.FromRgb(0, 128, 128);
00216 /// <summary>
00217 /// DarkCyan #008B8B
00218 /// </summary>
00219         public static Colour DarkCyan = Colour.FromRgb(0, 139, 139);
00220 /// <summary>
00221 /// DeepSkyBlue #00BFFF
00222 /// </summary>
00223         public static Colour DeepSkyBlue = Colour.FromRgb(0, 191, 255);
00224 /// <summary>
00225 /// DarkTurquoise #00CED1
00226 /// </summary>
00227         public static Colour DarkTurquoise = Colour.FromRgb(0, 206, 209);
00228 /// <summary>
00229 /// MediumSpringGreen #00FA9A
00230 /// </summary>
00231         public static Colour MediumSpringGreen = Colour.FromRgb(0, 250, 154);
00232 /// <summary>
00233 /// Lime #00FF00
00234 /// </summary>
00235         public static Colour Lime = Colour.FromRgb(0, 255, 0);
00236 /// <summary>
00237 /// SpringGreen #00FF7F
00238 /// </summary>
00239         public static Colour SpringGreen = Colour.FromRgb(0, 255, 127);
00240 /// <summary>
00241 /// Aqua #00FFFF
00242 /// </summary>
00243         public static Colour Aqua = Colour.FromRgb(0, 255, 255);
00244 /// <summary>
00245 /// Cyan #00FFFF
00246 /// </summary>
00247         public static Colour Cyan = Colour.FromRgb(0, 255, 255);
00248 /// <summary>
00249 /// MidnightBlue #191970
00250 /// </summary>
00251         public static Colour MidnightBlue = Colour.FromRgb(25, 25, 112);
00252 /// <summary>
00253 /// DodgerBlue #1E90FF
00254 /// </summary>
00255         public static Colour DodgerBlue = Colour.FromRgb(30, 144, 255);
00256 /// <summary>
00257 /// LightSeaGreen #20B2AA
00258 /// </summary>
00259         public static Colour LightSeaGreen = Colour.FromRgb(32, 178, 170);
00260 /// <summary>
00261 /// ForestGreen #228B22
00262 /// </summary>
00263         public static Colour ForestGreen = Colour.FromRgb(34, 139, 34);
00264 /// <summary>
00265 /// SeaGreen #2E8B57
00266 /// </summary>
00267         public static Colour SeaGreen = Colour.FromRgb(46, 139, 87);
00268 /// <summary>
00269 /// DarkSlateGray #2F4F4F
00270 /// </summary>
00271         public static Colour DarkSlateGray = Colour.FromRgb(47, 79, 79);
00272 /// <summary>
00273 /// DarkSlateGrey #2F4F4F
```

```
00274 /// </summary>
00275         public static Colour DarkSlateGrey = Colour.FromRgb(47, 79, 79);
00276 /// <summary>
00277 /// LimeGreen #32CD32
00278 /// </summary>
00279         public static Colour LimeGreen = Colour.FromRgb(50, 205, 50);
00280 /// <summary>
00281 /// MediumSeaGreen #3CB371
00282 /// </summary>
00283         public static Colour MediumSeaGreen = Colour.FromRgb(60, 179, 113);
00284 /// <summary>
00285 /// Turquoise #40E0D0
00286 /// </summary>
00287         public static Colour Turquoise = Colour.FromRgb(64, 224, 208);
00288 /// <summary>
00289 /// RoyalBlue #4169E1
00290 /// </summary>
00291         public static Colour RoyalBlue = Colour.FromRgb(65, 105, 225);
00292 /// <summary>
00293 /// SteelBlue #4682B4
00294 /// </summary>
00295         public static Colour SteelBlue = Colour.FromRgb(70, 130, 180);
00296 /// <summary>
00297 /// DarkSlateBlue #483D8B
00298 /// </summary>
00299         public static Colour DarkSlateBlue = Colour.FromRgb(72, 61, 139);
00300 /// <summary>
00301 /// MediumTurquoise #48D1CC
00302 /// </summary>
00303         public static Colour MediumTurquoise = Colour.FromRgb(72, 209, 204);
00304 /// <summary>
00305 /// Indigo #4B0082
00306 /// </summary>
00307         public static Colour Indigo = Colour.FromRgb(75, 0, 130);
00308 /// <summary>
00309 /// DarkOliveGreen #556B2F
00310 /// </summary>
00311         public static Colour DarkOliveGreen = Colour.FromRgb(85, 107, 47);
00312 /// <summary>
00313 /// CadetBlue #5F9EA0
00314 /// </summary>
00315         public static Colour CadetBlue = Colour.FromRgb(95, 158, 160);
00316 /// <summary>
00317 /// CornflowerBlue #6495ED
00318 /// </summary>
00319         public static Colour CornflowerBlue = Colour.FromRgb(100, 149, 237);
00320 /// <summary>
00321 /// RebeccaPurple #663399
00322 /// </summary>
00323         public static Colour RebeccaPurple = Colour.FromRgb(102, 51, 153);
00324 /// <summary>
00325 /// MediumAquaMarine #66CDAA
00326 /// </summary>
00327         public static Colour MediumAquaMarine = Colour.FromRgb(102, 205, 170);
00328 /// <summary>
00329 /// DimGray #696969
00330 /// </summary>
00331         public static Colour DimGray = Colour.FromRgb(105, 105, 105);
00332 /// <summary>
00333 /// DimGrey #696969
00334 /// </summary>
00335         public static Colour DimGrey = Colour.FromRgb(105, 105, 105);
00336 /// <summary>
00337 /// SlateBlue #6A5ACD
00338 /// </summary>
00339         public static Colour SlateBlue = Colour.FromRgb(106, 90, 205);
00340 /// <summary>
00341 /// OliveDrab #6B8E23
00342 /// </summary>
00343         public static Colour OliveDrab = Colour.FromRgb(107, 142, 35);
00344 /// <summary>
00345 /// SlateGray #708090
00346 /// </summary>
00347         public static Colour SlateGray = Colour.FromRgb(112, 128, 144);
00348 /// <summary>
00349 /// SlateGrey #708090
00350 /// </summary>
00351         public static Colour SlateGrey = Colour.FromRgb(112, 128, 144);
00352 /// <summary>
00353 /// LightSlateGray #778899
00354 /// </summary>
00355         public static Colour LightSlateGray = Colour.FromRgb(119, 136, 153);
00356 /// <summary>
00357 /// LightSlateGrey #778899
00358 /// </summary>
00359         public static Colour LightSlateGrey = Colour.FromRgb(119, 136, 153);
00360 /// <summary>
```

```
00361 /// MediumSlateBlue #7B68EE
00362 /// </summary>
00363         public static Colour MediumSlateBlue = Colour.FromRgb(123, 104, 238);
00364 /// <summary>
00365 /// LawnGreen #7CFC00
00366 /// </summary>
00367         public static Colour LawnGreen = Colour.FromRgb(124, 252, 0);
00368 /// <summary>
00369 /// Chartreuse #7FFF00
00370 /// </summary>
00371         public static Colour Chartreuse = Colour.FromRgb(127, 255, 0);
00372 /// <summary>
00373 /// Aquamarine #7FFFD4
00374 /// </summary>
00375         public static Colour Aquamarine = Colour.FromRgb(127, 255, 212);
00376 /// <summary>
00377 /// Maroon #800000
00378 /// </summary>
00379         public static Colour Maroon = Colour.FromRgb(128, 0, 0);
00380 /// <summary>
00381 /// Purple #800080
00382 /// </summary>
00383         public static Colour Purple = Colour.FromRgb(128, 0, 128);
00384 /// <summary>
00385 /// Olive #808000
00386 /// </summary>
00387         public static Colour Olive = Colour.FromRgb(128, 128, 0);
00388 /// <summary>
00389 /// Gray #808080
00390 /// </summary>
00391         public static Colour Gray = Colour.FromRgb(128, 128, 128);
00392 /// <summary>
00393 /// Grey #808080
00394 /// </summary>
00395         public static Colour Grey = Colour.FromRgb(128, 128, 128);
00396 /// <summary>
00397 /// SkyBlue #87CEEB
00398 /// </summary>
00399         public static Colour SkyBlue = Colour.FromRgb(135, 206, 235);
00400 /// <summary>
00401 /// LightSkyBlue #87CEFA
00402 /// </summary>
00403         public static Colour LightSkyBlue = Colour.FromRgb(135, 206, 250);
00404 /// <summary>
00405 /// BlueViolet #8A2BE2
00406 /// </summary>
00407         public static Colour BlueViolet = Colour.FromRgb(138, 43, 226);
00408 /// <summary>
00409 /// DarkRed #8B0000
00410 /// </summary>
00411         public static Colour DarkRed = Colour.FromRgb(139, 0, 0);
00412 /// <summary>
00413 /// DarkMagenta #8B008B
00414 /// </summary>
00415         public static Colour DarkMagenta = Colour.FromRgb(139, 0, 139);
00416 /// <summary>
00417 /// SaddleBrown #8B4513
00418 /// </summary>
00419         public static Colour SaddleBrown = Colour.FromRgb(139, 69, 19);
00420 /// <summary>
00421 /// DarkSeaGreen #8FBC8F
00422 /// </summary>
00423         public static Colour DarkSeaGreen = Colour.FromRgb(143, 188, 143);
00424 /// <summary>
00425 /// LightGreen #90EE90
00426 /// </summary>
00427         public static Colour LightGreen = Colour.FromRgb(144, 238, 144);
00428 /// <summary>
00429 /// MediumPurple #9370DB
00430 /// </summary>
00431         public static Colour MediumPurple = Colour.FromRgb(147, 112, 219);
00432 /// <summary>
00433 /// DarkViolet #9400D3
00434 /// </summary>
00435         public static Colour DarkViolet = Colour.FromRgb(148, 0, 211);
00436 /// <summary>
00437 /// PaleGreen #98FB98
00438 /// </summary>
00439         public static Colour PaleGreen = Colour.FromRgb(152, 251, 152);
00440 /// <summary>
00441 /// DarkOrchid #9932CC
00442 /// </summary>
00443         public static Colour DarkOrchid = Colour.FromRgb(153, 50, 204);
00444 /// <summary>
00445 /// YellowGreen #9ACD32
00446 /// </summary>
00447         public static Colour YellowGreen = Colour.FromRgb(154, 205, 50);
```

```
00448 /// <summary>
00449 /// Sienna #A0522D
00450 /// </summary>
00451         public static Colour Sienna = Colour.FromRgb(160, 82, 45);
00452 /// <summary>
00453 /// Brown #A52A2A
00454 /// </summary>
00455         public static Colour Brown = Colour.FromRgb(165, 42, 42);
00456 /// <summary>
00457 /// DarkGray #A9A9A9
00458 /// </summary>
00459         public static Colour DarkGray = Colour.FromRgb(169, 169, 169);
00460 /// <summary>
00461 /// DarkGrey #A9A9A9
00462 /// </summary>
00463         public static Colour DarkGrey = Colour.FromRgb(169, 169, 169);
00464 /// <summary>
00465 /// LightBlue #ADD8E6
00466 /// </summary>
00467         public static Colour LightBlue = Colour.FromRgb(173, 216, 230);
00468 /// <summary>
00469 /// GreenYellow #ADFF2F
00470 /// </summary>
00471         public static Colour GreenYellow = Colour.FromRgb(173, 255, 47);
00472 /// <summary>
00473 /// PaleTurquoise #AFEEEE
00474 /// </summary>
00475         public static Colour PaleTurquoise = Colour.FromRgb(175, 238, 238);
00476 /// <summary>
00477 /// LightSteelBlue #B0C4DE
00478 /// </summary>
00479         public static Colour LightSteelBlue = Colour.FromRgb(176, 196, 222);
00480 /// <summary>
00481 /// PowderBlue #B0E0E6
00482 /// </summary>
00483         public static Colour PowderBlue = Colour.FromRgb(176, 224, 230);
00484 /// <summary>
00485 /// FireBrick #B22222
00486 /// </summary>
00487         public static Colour FireBrick = Colour.FromRgb(178, 34, 34);
00488 /// <summary>
00489 /// DarkGoldenRod #B8860B
00490 /// </summary>
00491         public static Colour DarkGoldenRod = Colour.FromRgb(184, 134, 11);
00492 /// <summary>
00493 /// MediumOrchid #BA55D3
00494 /// </summary>
00495         public static Colour MediumOrchid = Colour.FromRgb(186, 85, 211);
00496 /// <summary>
00497 /// RosyBrown #BC8F8F
00498 /// </summary>
00499         public static Colour RosyBrown = Colour.FromRgb(188, 143, 143);
00500 /// <summary>
00501 /// DarkKhaki #BDB76B
00502 /// </summary>
00503         public static Colour DarkKhaki = Colour.FromRgb(189, 183, 107);
00504 /// <summary>
00505 /// Silver #C0C0C0
00506 /// </summary>
00507         public static Colour Silver = Colour.FromRgb(192, 192, 192);
00508 /// <summary>
00509 /// MediumVioletRed #C71585
00510 /// </summary>
00511         public static Colour MediumVioletRed = Colour.FromRgb(199, 21, 133);
00512 /// <summary>
00513 /// IndianRed #CD5C5C
00514 /// </summary>
00515         public static Colour IndianRed = Colour.FromRgb(205, 92, 92);
00516 /// <summary>
00517 /// Peru #CD853F
00518 /// </summary>
00519         public static Colour Peru = Colour.FromRgb(205, 133, 63);
00520 /// <summary>
00521 /// Chocolate #D2691E
00522 /// </summary>
00523         public static Colour Chocolate = Colour.FromRgb(210, 105, 30);
00524 /// <summary>
00525 /// Tan #D2B48C
00526 /// </summary>
00527         public static Colour Tan = Colour.FromRgb(210, 180, 140);
00528 /// <summary>
00529 /// LightGray #D3D3D3
00530 /// </summary>
00531         public static Colour LightGray = Colour.FromRgb(211, 211, 211);
00532 /// <summary>
00533 /// LightGrey #D3D3D3
00534 /// </summary>
```

```
00535         public static Colour LightGrey = Colour.FromRgb(211, 211, 211);
00536 /// <summary>
00537 /// Thistle #D8BFD8
00538 /// </summary>
00539         public static Colour Thistle = Colour.FromRgb(216, 191, 216);
00540 /// <summary>
00541 /// Orchid #DA70D6
00542 /// </summary>
00543         public static Colour Orchid = Colour.FromRgb(218, 112, 214);
00544 /// <summary>
00545 /// GoldenRod #DAA520
00546 /// </summary>
00547         public static Colour GoldenRod = Colour.FromRgb(218, 165, 32);
00548 /// <summary>
00549 /// PaleVioletRed #DB7093
00550 /// </summary>
00551         public static Colour PaleVioletRed = Colour.FromRgb(219, 112, 147);
00552 /// <summary>
00553 /// Crimson #DC143C
00554 /// </summary>
00555         public static Colour Crimson = Colour.FromRgb(220, 20, 60);
00556 /// <summary>
00557 /// Gainsboro #DCDCDC
00558 /// </summary>
00559         public static Colour Gainsboro = Colour.FromRgb(220, 220, 220);
00560 /// <summary>
00561 /// Plum #DDA0DD
00562 /// </summary>
00563         public static Colour Plum = Colour.FromRgb(221, 160, 221);
00564 /// <summary>
00565 /// BurlyWood #DEB887
00566 /// </summary>
00567         public static Colour BurlyWood = Colour.FromRgb(222, 184, 135);
00568 /// <summary>
00569 /// LightCyan #E0FFFF
00570 /// </summary>
00571         public static Colour LightCyan = Colour.FromRgb(224, 255, 255);
00572 /// <summary>
00573 /// Lavender #E6E6FA
00574 /// </summary>
00575         public static Colour Lavender = Colour.FromRgb(230, 230, 250);
00576 /// <summary>
00577 /// DarkSalmon #E9967A
00578 /// </summary>
00579         public static Colour DarkSalmon = Colour.FromRgb(233, 150, 122);
00580 /// <summary>
00581 /// Violet #EE82EE
00582 /// </summary>
00583         public static Colour Violet = Colour.FromRgb(238, 130, 238);
00584 /// <summary>
00585 /// PaleGoldenRod #EEE8AA
00586 /// </summary>
00587         public static Colour PaleGoldenRod = Colour.FromRgb(238, 232, 170);
00588 /// <summary>
00589 /// LightCoral #F08080
00590 /// </summary>
00591         public static Colour LightCoral = Colour.FromRgb(240, 128, 128);
00592 /// <summary>
00593 /// Khaki #F0E68C
00594 /// </summary>
00595         public static Colour Khaki = Colour.FromRgb(240, 230, 140);
00596 /// <summary>
00597 /// AliceBlue #F0F8FF
00598 /// </summary>
00599         public static Colour AliceBlue = Colour.FromRgb(240, 248, 255);
00600 /// <summary>
00601 /// HoneyDew #F0FFF0
00602 /// </summary>
00603         public static Colour HoneyDew = Colour.FromRgb(240, 255, 240);
00604 /// <summary>
00605 /// Azure #F0FFFF
00606 /// </summary>
00607         public static Colour Azure = Colour.FromRgb(240, 255, 255);
00608 /// <summary>
00609 /// SandyBrown #F4A460
00610 /// </summary>
00611         public static Colour SandyBrown = Colour.FromRgb(244, 164, 96);
00612 /// <summary>
00613 /// Wheat #F5DEB3
00614 /// </summary>
00615         public static Colour Wheat = Colour.FromRgb(245, 222, 179);
00616 /// <summary>
00617 /// Beige #F5F5DC
00618 /// </summary>
00619         public static Colour Beige = Colour.FromRgb(245, 245, 220);
00620 /// <summary>
00621 /// WhiteSmoke #F5F5F5
```

```
00622 /// </summary>
00623         public static Colour WhiteSmoke = Colour.FromRgb(245, 245, 245);
00624 /// <summary>
00625 /// MintCream #F5FFFA
00626 /// </summary>
00627         public static Colour MintCream = Colour.FromRgb(245, 255, 250);
00628 /// <summary>
00629 /// GhostWhite #F8F8FF
00630 /// </summary>
00631         public static Colour GhostWhite = Colour.FromRgb(248, 248, 255);
00632 /// <summary>
00633 /// Salmon #FA8072
00634 /// </summary>
00635         public static Colour Salmon = Colour.FromRgb(250, 128, 114);
00636 /// <summary>
00637 /// AntiqueWhite #FAEBD7
00638 /// </summary>
00639         public static Colour AntiqueWhite = Colour.FromRgb(250, 235, 215);
00640 /// <summary>
00641 /// Linen #FAF0E6
00642 /// </summary>
00643         public static Colour Linen = Colour.FromRgb(250, 240, 230);
00644 /// <summary>
00645 /// LightGoldenRodYellow #FAFAD2
00646 /// </summary>
00647         public static Colour LightGoldenRodYellow = Colour.FromRgb(250, 250, 210);
00648 /// <summary>
00649 /// OldLace #FDF5E6
00650 /// </summary>
00651         public static Colour OldLace = Colour.FromRgb(253, 245, 230);
00652 /// <summary>
00653 /// Red #FF0000
00654 /// </summary>
00655         public static Colour Red = Colour.FromRgb(255, 0, 0);
00656 /// <summary>
00657 /// Fuchsia #FF00FF
00658 /// </summary>
00659         public static Colour Fuchsia = Colour.FromRgb(255, 0, 255);
00660 /// <summary>
00661 /// Magenta #FF00FF
00662 /// </summary>
00663         public static Colour Magenta = Colour.FromRgb(255, 0, 255);
00664 /// <summary>
00665 /// DeepPink #FF1493
00666 /// </summary>
00667         public static Colour DeepPink = Colour.FromRgb(255, 20, 147);
00668 /// <summary>
00669 /// OrangeRed #FF4500
00670 /// </summary>
00671         public static Colour OrangeRed = Colour.FromRgb(255, 69, 0);
00672 /// <summary>
00673 /// Tomato #FF6347
00674 /// </summary>
00675         public static Colour Tomato = Colour.FromRgb(255, 99, 71);
00676 /// <summary>
00677 /// HotPink #FF69B4
00678 /// </summary>
00679         public static Colour HotPink = Colour.FromRgb(255, 105, 180);
00680 /// <summary>
00681 /// Coral #FF7F50
00682 /// </summary>
00683         public static Colour Coral = Colour.FromRgb(255, 127, 80);
00684 /// <summary>
00685 /// DarkOrange #FF8C00
00686 /// </summary>
00687         public static Colour DarkOrange = Colour.FromRgb(255, 140, 0);
00688 /// <summary>
00689 /// LightSalmon #FFA07A
00690 /// </summary>
00691         public static Colour LightSalmon = Colour.FromRgb(255, 160, 122);
00692 /// <summary>
00693 /// Orange #FFA500
00694 /// </summary>
00695         public static Colour Orange = Colour.FromRgb(255, 165, 0);
00696 /// <summary>
00697 /// LightPink #FFB6C1
00698 /// </summary>
00699         public static Colour LightPink = Colour.FromRgb(255, 182, 193);
00700 /// <summary>
00701 /// Pink #FFC0CB
00702 /// </summary>
00703         public static Colour Pink = Colour.FromRgb(255, 192, 203);
00704 /// <summary>
00705 /// Gold #FFD700
00706 /// </summary>
00707         public static Colour Gold = Colour.FromRgb(255, 215, 0);
00708 /// <summary>
```

```
00709 /// PeachPuff #FFDAB9
00710 /// </summary>
00711         public static Colour PeachPuff = Colour.FromRgb(255, 218, 185);
00712 /// <summary>
00713 /// NavajoWhite #FFDEAD
00714 /// </summary>
00715         public static Colour NavajoWhite = Colour.FromRgb(255, 222, 173);
00716 /// <summary>
00717 /// Moccasin #FFE4B5
00718 /// </summary>
00719         public static Colour Moccasin = Colour.FromRgb(255, 228, 181);
00720 /// <summary>
00721 /// Bisque #FFE4C4
00722 /// </summary>
00723         public static Colour Bisque = Colour.FromRgb(255, 228, 196);
00724 /// <summary>
00725 /// MistyRose #FFE4E1
00726 /// </summary>
00727         public static Colour MistyRose = Colour.FromRgb(255, 228, 225);
00728 /// <summary>
00729 /// BlanchedAlmond #FFEBCD
00730 /// </summary>
00731         public static Colour BlanchedAlmond = Colour.FromRgb(255, 235, 205);
00732 /// <summary>
00733 /// PapayaWhip #FFEFD5
00734 /// </summary>
00735         public static Colour PapayaWhip = Colour.FromRgb(255, 239, 213);
00736 /// <summary>
00737 /// LavenderBlush #FFF0F5
00738 /// </summary>
00739         public static Colour LavenderBlush = Colour.FromRgb(255, 240, 245);
00740 /// <summary>
00741 /// SeaShell #FFF5EE
00742 /// </summary>
00743         public static Colour SeaShell = Colour.FromRgb(255, 245, 238);
00744 /// <summary>
00745 /// Cornsilk #FFF8DC
00746 /// </summary>
00747         public static Colour Cornsilk = Colour.FromRgb(255, 248, 220);
00748 /// <summary>
00749 /// LemonChiffon #FFFACD
00750 /// </summary>
00751         public static Colour LemonChiffon = Colour.FromRgb(255, 250, 205);
00752 /// <summary>
00753 /// FloralWhite #FFFAF0
00754 /// </summary>
00755         public static Colour FloralWhite = Colour.FromRgb(255, 250, 240);
00756 /// <summary>
00757 /// Snow #FFFAFA
00758 /// </summary>
00759         public static Colour Snow = Colour.FromRgb(255, 250, 250);
00760 /// <summary>
00761 /// Yellow #FFFF00
00762 /// </summary>
00763         public static Colour Yellow = Colour.FromRgb(255, 255, 0);
00764 /// <summary>
00765 /// LightYellow #FFFFE0
00766 /// </summary>
00767         public static Colour LightYellow = Colour.FromRgb(255, 255, 224);
00768 /// <summary>
00769 /// Ivory #FFFFF0
00770 /// </summary>
00771         public static Colour Ivory = Colour.FromRgb(255, 255, 240);
00772 /// <summary>
00773 /// White #FFFFFF
00774 /// </summary>
00775         public static Colour White = Colour.FromRgb(255, 255, 255);
00776     }
00777 }
```

8.93 TrueType.cs

```
00001 /*
00002 VectSharp - A light library for C# vector graphics.
00003 Copyright (C) 2020-2022 Giorgio Bianchini
00004
00005 This program is free software:  you can redistribute it and/or modify
00006 it under the terms of the GNU Lesser General Public License as published by
00007 the Free Software Foundation, version 3.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
```

```
00012 GNU Lesser General Public License for more details.
00013
00014 You should have received a copy of the GNU Lesser General Public License
00015 along with this program.  If not, see <https://www.gnu.org/licenses/>.
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Diagnostics.Contracts;
00021 using System.IO;
00022 using System.Linq;
00023 using System.Text;
00024
00025 namespace VectSharp
00026 {
00027 /// <summary>
00028 /// Represents a font file in TrueType format.  Reference:  http://stevehanov.ca/blog/?id=143,
      https://developer.apple.com/fonts/TrueType-Reference-Manual/,
      https://docs.microsoft.com/en-us/typography/opentype/spec/
00029 /// </summary>
00030     public class TrueTypeFile
00031     {
00032         private static readonly Dictionary<string, TrueTypeFile> FontCache = new Dictionary<string,
      TrueTypeFile>();
00033         private static readonly Dictionary<Stream, TrueTypeFile> StreamFontCache = new
      Dictionary<Stream, TrueTypeFile>();
00034         private static object FontCacheLock = new object();
00035         internal uint ScalarType { get; }
00036         internal ushort NumTables { get; }
00037         internal ushort SearchRange { get; }
00038         internal ushort EntrySelector { get; }
00039         internal ushort RangeShift { get; }
00040         internal Dictionary<string, TrueTypeTableOffset> TableOffsets { get; } = new
      Dictionary<string, TrueTypeTableOffset>();
00041         internal Dictionary<string, ITrueTypeTable> Tables { get; } = new Dictionary<string,
      ITrueTypeTable>();
00042
00043
00044 /// <summary>
00045 /// A stream pointing to the TrueType file source (either on disk or in memory).  Never dispose this
      stream directly; if you really need to, call <see cref="Destroy"/> instead.
00046 /// </summary>
00047         public Stream FontStream { get; }
00048
00049 /// <summary>
00050 /// Remove this TrueType file from the cache, clear the tables and release the <see
      cref="FontStream"/>.
00051 /// Only call this when the actual file that was used to create this object needs to be changed!
00052 /// </summary>
00053         public void Destroy()
00054         {
00055             string keyString = null;
00056
00057
00058             lock (FontCacheLock)
00059             {
00060                 foreach (KeyValuePair<string, TrueTypeFile> kvp in FontCache)
00061                 {
00062                     if (kvp.Value == this)
00063                     {
00064                         keyString = kvp.Key;
00065                         break;
00066                     }
00067                 }
00068
00069
00070                 if (!string.IsNullOrEmpty(keyString))
00071                 {
00072                     FontCache.Remove(keyString);
00073                 }
00074
00075                 Stream keyStream = null;
00076
00077                 foreach (KeyValuePair<Stream, TrueTypeFile> kvp in StreamFontCache)
00078                 {
00079                     if (kvp.Value == this)
00080                     {
00081                         keyStream = kvp.Key;
00082                     }
00083                 }
00084
00085                 if (keyStream != null)
00086                 {
00087                     StreamFontCache.Remove(keyStream);
00088                 }
00089             }
00090
```

```
00091             this.FontStream.Dispose();
00092             this.TableOffsets.Clear();
00093             this.Tables.Clear();
00094         }
00095 
00096         internal TrueTypeFile(Dictionary<string, ITrueTypeTable> tables)
00097         {
00098             this.Tables = tables;
00099 
00100             this.ScalarType = (uint)65536;
00101             this.NumTables = (ushort)tables.Count;
00102             this.EntrySelector = (ushort)Math.Floor(Math.Log(tables.Count, 2));
00103             this.SearchRange = (ushort)(16 * (1 << this.EntrySelector));
00104             this.RangeShift = (ushort)(tables.Count * 16 - this.SearchRange);
00105 
00106             uint offset = 12 + 16 * (uint)tables.Count;
00107 
00108             uint fontChecksum = this.ScalarType;
00109             fontChecksum += ((uint)this.NumTables << 16) + (uint)this.SearchRange;
00110             fontChecksum += ((uint)this.EntrySelector << 16) + (uint)this.RangeShift;
00111 
00112             //For some reason, OpenType Sanitizer (https://github.com/khaledhosny/ots), integrated
    into Chromium and Firefox, complains if the tables are ordered differently.
00113             List<string> tableOrdering = new List<string>()
00114             {
00115                 "OS/2", "cmap", "glyf", "head", "hhea", "hmtx", "loca", "maxp", "name", "post"
00116             };
00117 
00118             Dictionary<string, byte[]> allBytes = new Dictionary<string, byte[]>();
00119 
00120             foreach (KeyValuePair<string, ITrueTypeTable> kvp in from el in Tables let ind =
    tableOrdering.IndexOf(el.Key) orderby (ind >= 0 ? ind : tableOrdering.Count) ascending select el)
00121             {
00122                 if (kvp.Key == "head")
00123                 {
00124                     ((TrueTypeHeadTable)kvp.Value).ChecksumAdjustment = 0;
00125                 }
00126 
00127                 byte[] tableBytes = kvp.Value.GetBytes();
00128                 allBytes.Add(kvp.Key, tableBytes);
00129                 int length = tableBytes.Length;
00130 
00131                 uint checksum = 0;
00132 
00133                 for (int i = 0; i < tableBytes.Length; i += 4)
00134                 {
00135                     byte b1 = tableBytes[i];
00136                     byte b2 = (i + 1 < tableBytes.Length) ? tableBytes[i + 1] : (byte)0;
00137                     byte b3 = (i + 2 < tableBytes.Length) ? tableBytes[i + 2] : (byte)0;
00138                     byte b4 = (i + 3 < tableBytes.Length) ? tableBytes[i + 3] : (byte)0;
00139 
00140                     checksum += (uint)((b1 << 24) + (b2 << 16) + (b3 << 8) + b4);
00141                 }
00142 
00143                 TableOffsets.Add(kvp.Key, new TrueTypeTableOffset(checksum, offset, (uint)length));
00144 
00145                 fontChecksum += checksum;
00146 
00147                 fontChecksum += (uint)(((byte)kvp.Key[0] << 24) + ((byte)kvp.Key[1] << 16) +
    ((byte)kvp.Key[2] << 8) + (byte)kvp.Key[3]);
00148                 fontChecksum += offset;
00149                 fontChecksum += checksum;
00150                 fontChecksum += (uint)length;
00151 
00152                 offset += (uint)length;
00153 
00154                 switch (length % 4)
00155                 {
00156                     case 1:
00157                         offset += 3;
00158                         break;
00159                     case 2:
00160                         offset += 2;
00161                         break;
00162                     case 3:
00163                         offset++;
00164                         break;
00165                 }
00166             }
00167 
00168             ((TrueTypeHeadTable)this.Tables["head"]).ChecksumAdjustment = 0xB1B0AFBA - fontChecksum;
00169 
00170 
00171             FontStream = new MemoryStream();
00172             FontStream.WriteUInt(this.ScalarType);
00173             FontStream.WriteUShort(this.NumTables);
00174             FontStream.WriteUShort(this.SearchRange);
```

```
00175             FontStream.WriteUShort(this.EntrySelector);
00176             FontStream.WriteUShort(this.RangeShift);
00177 
00178             foreach (KeyValuePair<string, ITrueTypeTable> kvp in from el in Tables let ind =
      tableOrdering.IndexOf(el.Key) orderby (ind >= 0 ?  ind :  tableOrdering.Count) ascending select el)
00179             {
00180                 FontStream.Write(Encoding.ASCII.GetBytes(kvp.Key), 0, 4);
00181                 FontStream.WriteUInt(this.TableOffsets[kvp.Key].Checksum);
00182                 FontStream.WriteUInt(this.TableOffsets[kvp.Key].Offset);
00183                 FontStream.WriteUInt(this.TableOffsets[kvp.Key].Length);
00184             }
00185 
00186             foreach (KeyValuePair<string, ITrueTypeTable> kvp in from el in Tables let ind =
      tableOrdering.IndexOf(el.Key) orderby (ind >= 0 ?  ind :  tableOrdering.Count) ascending select el)
00187             {
00188                 FontStream.Write(allBytes[kvp.Key], 0, allBytes[kvp.Key].Length);
00189                 switch (allBytes[kvp.Key].Length % 4)
00190                 {
00191                     case 1:
00192                         FontStream.WriteByte(0);
00193                         FontStream.WriteByte(0);
00194                         FontStream.WriteByte(0);
00195                         break;
00196                     case 2:
00197                         FontStream.WriteByte(0);
00198                         FontStream.WriteByte(0);
00199                         break;
00200                     case 3:
00201                         FontStream.WriteByte(0);
00202                         break;
00203                 }
00204             }
00205 
00206             FontStream.Seek(0, SeekOrigin.Begin);
00207 
00208             BuildGlyphCache();
00209         }
00210 
00211         internal static TrueTypeFile CreateTrueTypeFile(string fileName)
00212         {
00213             lock (FontCacheLock)
00214             {
00215                 if (!FontCache.ContainsKey(fileName))
00216                 {
00217                     FontCache.Add(fileName, new TrueTypeFile(fileName));
00218                 }
00219 
00220                 return FontCache[fileName];
00221             }
00222         }
00223 
00224         internal static TrueTypeFile CreateTrueTypeFile(Stream fontStream)
00225         {
00226             lock (FontCacheLock)
00227             {
00228                 if (!StreamFontCache.ContainsKey(fontStream))
00229                 {
00230                     StreamFontCache.Add(fontStream, new TrueTypeFile(fontStream));
00231                 }
00232 
00233                 return StreamFontCache[fontStream];
00234             }
00235         }
00236 
00237         private TrueTypeFile(string fileName) :  this(new FileStream(fileName, FileMode.Open,
      FileAccess.Read, FileShare.Read))
00238         {
00239 
00240         }
00241 
00242         private TrueTypeFile(Stream fs)
00243         {
00244             FontStream = fs;
00245             this.ScalarType = fs.ReadUInt();
00246             this.NumTables = fs.ReadUShort();
00247             this.SearchRange = fs.ReadUShort();
00248             this.EntrySelector = fs.ReadUShort();
00249             this.RangeShift = fs.ReadUShort();
00250 
00251             for (int i = 0; i < NumTables; i++)
00252             {
00253                 this.TableOffsets.Add(fs.ReadString(4), new TrueTypeTableOffset(fs.ReadUInt(),
      fs.ReadUInt(), fs.ReadUInt()));
00254             }
00255 
00256             fs.Seek(this.TableOffsets["head"].Offset, SeekOrigin.Begin);
00257 
```

```
00258             Tables.Add("head", new TrueTypeHeadTable()
00259             {
00260                 Version = fs.ReadFixed(),
00261                 FontRevision = fs.ReadFixed(),
00262                 ChecksumAdjustment = fs.ReadUInt(),
00263                 MagicNumber = fs.ReadUInt(),
00264                 Flags = fs.ReadUShort(),
00265                 UnitsPerEm = fs.ReadUShort(),
00266                 Created = fs.ReadDate(),
00267                 Modified = fs.ReadDate(),
00268                 XMin = fs.ReadShort(),
00269                 YMin = fs.ReadShort(),
00270                 XMax = fs.ReadShort(),
00271                 YMax = fs.ReadShort(),
00272                 MacStyle = fs.ReadUShort(),
00273                 LowestRecPPEM = fs.ReadUShort(),
00274                 FontDirectionInt = fs.ReadShort(),
00275                 IndexToLocFormat = fs.ReadShort(),
00276                 GlyphDataFormat = fs.ReadShort()
00277             });
00278
00279             fs.Seek(this.TableOffsets["hhea"].Offset, SeekOrigin.Begin);
00280
00281             Tables.Add("hhea", new TrueTypeHHeaTable()
00282             {
00283                 Version = fs.ReadFixed(),
00284                 Ascent = fs.ReadShort(),
00285                 Descent = fs.ReadShort(),
00286                 LineGap = fs.ReadShort(),
00287                 AdvanceWidthMax = fs.ReadUShort(),
00288                 MinLeftSideBearing = fs.ReadShort(),
00289                 MinRightSideBearing = fs.ReadShort(),
00290                 XMaxExtent = fs.ReadShort(),
00291                 CaretSlopeRise = fs.ReadShort(),
00292                 CaretSlopeRun = fs.ReadShort(),
00293                 CaretOffset = fs.ReadShort()
00294             });
00295
00296             fs.ReadShort();
00297             fs.ReadShort();
00298             fs.ReadShort();
00299             fs.ReadShort();
00300
00301             ((TrueTypeHHeaTable)Tables["hhea"]).MetricDataFormat = fs.ReadShort();
00302             ((TrueTypeHHeaTable)Tables["hhea"]).NumOfLongHorMetrics = fs.ReadUShort();
00303
00304             fs.Seek(this.TableOffsets["maxp"].Offset, SeekOrigin.Begin);
00305
00306             Tables.Add("maxp", new TrueTypeMaxpTable()
00307             {
00308                 Version = fs.ReadFixed(),
00309                 NumGlyphs = fs.ReadUShort()
00310             });
00311
00312             if (((TrueTypeMaxpTable)Tables["maxp"]).Version.Bits == 65536)
00313             {
00314                 ((TrueTypeMaxpTable)Tables["maxp"]).MaxPoints = fs.ReadUShort();
00315                 ((TrueTypeMaxpTable)Tables["maxp"]).MaxContours = fs.ReadUShort();
00316                 ((TrueTypeMaxpTable)Tables["maxp"]).MaxComponentPoints = fs.ReadUShort();
00317                 ((TrueTypeMaxpTable)Tables["maxp"]).MaxComponentContours = fs.ReadUShort();
00318                 ((TrueTypeMaxpTable)Tables["maxp"]).MaxZones = fs.ReadUShort();
00319                 ((TrueTypeMaxpTable)Tables["maxp"]).MaxTwilightPoints = fs.ReadUShort();
00320                 ((TrueTypeMaxpTable)Tables["maxp"]).MaxStorage = fs.ReadUShort();
00321                 ((TrueTypeMaxpTable)Tables["maxp"]).MaxFunctionDefs = fs.ReadUShort();
00322                 ((TrueTypeMaxpTable)Tables["maxp"]).MaxInstructionDefs = fs.ReadUShort();
00323                 ((TrueTypeMaxpTable)Tables["maxp"]).MaxStackElements = fs.ReadUShort();
00324                 ((TrueTypeMaxpTable)Tables["maxp"]).MaxSizeOfInstructions = fs.ReadUShort();
00325                 ((TrueTypeMaxpTable)Tables["maxp"]).MaxComponentElements = fs.ReadUShort();
00326                 ((TrueTypeMaxpTable)Tables["maxp"]).MaxComponentDepth = fs.ReadUShort();
00327             }
00328
00329             int totalGlyphs = ((TrueTypeMaxpTable)Tables["maxp"]).NumGlyphs;
00330             int numMetrics = ((TrueTypeHHeaTable)Tables["hhea"]).NumOfLongHorMetrics;
00331
00332             fs.Seek(this.TableOffsets["hmtx"].Offset, SeekOrigin.Begin);
00333             Tables.Add("hmtx", new TrueTypeHmtxTable() { HMetrics = new LongHorFixed[numMetrics],
    LeftSideBearing = new short[totalGlyphs - numMetrics] });
00334
00335             for (int i = 0; i < numMetrics; i++)
00336             {
00337                 ((TrueTypeHmtxTable)Tables["hmtx"]).HMetrics[i] = new
    TrueTypeFile.LongHorFixed(fs.ReadUShort(), fs.ReadShort());
00338             }
00339
00340             for (int i = 0; i < totalGlyphs - numMetrics; i++)
00341             {
00342                 ((TrueTypeHmtxTable)Tables["hmtx"]).LeftSideBearing[i] = fs.ReadShort();
```

```
00343             }
00344
00345             fs.Seek(this.TableOffsets["cmap"].Offset, SeekOrigin.Begin);
00346             Tables.Add("cmap", new TrueTypeCmapTable() { Version = fs.ReadUShort(), NumberSubTables =
      fs.ReadUShort() });
00347
00348             ((TrueTypeCmapTable)Tables["cmap"]).SubTables = new
      CmapSubTable[((TrueTypeCmapTable)Tables["cmap"]).NumberSubTables];
00349             ((TrueTypeCmapTable)Tables["cmap"]).ActualCmapTables = new
      ICmapTable[((TrueTypeCmapTable)Tables["cmap"]).NumberSubTables];
00350
00351             for (int i = 0; i < ((TrueTypeCmapTable)Tables["cmap"]).NumberSubTables; i++)
00352             {
00353                 ((TrueTypeCmapTable)Tables["cmap"]).SubTables[i] = new
      TrueTypeFile.CmapSubTable(fs.ReadUShort(), fs.ReadUShort(), fs.ReadUInt());
00354             }
00355
00356
00357             for (int i = 0; i < ((TrueTypeCmapTable)Tables["cmap"]).NumberSubTables; i++)
00358             {
00359                 fs.Seek(((TrueTypeCmapTable)Tables["cmap"]).SubTables[i].Offset +
      TableOffsets["cmap"].Offset, SeekOrigin.Begin);
00360
00361                 ushort format = fs.ReadUShort();
00362                 ushort length = fs.ReadUShort();
00363                 ushort language = fs.ReadUShort();
00364
00365                 if (format == 0)
00366                 {
00367                     byte[] glyphIndexArray = new byte[256];
00368                     fs.Read(glyphIndexArray, 0, 256);
00369                     ((TrueTypeCmapTable)Tables["cmap"]).ActualCmapTables[i] = new CmapTable0(format,
      length, language, glyphIndexArray);
00370                 }
00371                 else if (format == 4)
00372                 {
00373                     ((TrueTypeCmapTable)Tables["cmap"]).ActualCmapTables[i] = new CmapTable4(format,
      length, language, fs);
00374                 }
00375             }
00376
00377             if (this.TableOffsets.ContainsKey("OS/2"))
00378             {
00379                 fs.Seek(this.TableOffsets["OS/2"].Offset, SeekOrigin.Begin);
00380
00381                 TrueTypeOS2Table os2 = new TrueTypeOS2Table()
00382                 {
00383                     Version = fs.ReadUShort(),
00384                     XAvgCharWidth = fs.ReadShort(),
00385                     UsWeightClass = fs.ReadUShort(),
00386                     UsWidthClass = fs.ReadUShort(),
00387                     FsType = fs.ReadShort(),
00388                     YSubscriptXSize = fs.ReadShort(),
00389                     YSubscriptYSize = fs.ReadShort(),
00390                     YSubscriptXOffset = fs.ReadShort(),
00391                     YSubscriptYOffset = fs.ReadShort(),
00392                     YSuperscriptXSize = fs.ReadShort(),
00393                     YSuperscriptYSize = fs.ReadShort(),
00394                     YSuperscriptXOffset = fs.ReadShort(),
00395                     YSuperscriptYOffset = fs.ReadShort(),
00396                     YStrikeoutSize = fs.ReadShort(),
00397                     YStrikeoutPosition = fs.ReadShort(),
00398                     SFamilyClass = (byte)fs.ReadByte(),
00399                     SFamilySubClass = (byte)fs.ReadByte(),
00400                     Panose = new TrueTypeOS2Table.PANOSE((byte)fs.ReadByte(), (byte)fs.ReadByte(),
      (byte)fs.ReadByte(), (byte)fs.ReadByte(), (byte)fs.ReadByte(), (byte)fs.ReadByte(),
      (byte)fs.ReadByte(), (byte)fs.ReadByte(), (byte)fs.ReadByte(), (byte)fs.ReadByte()),
00401                     UlUnicodeRange = new uint[] { fs.ReadUInt(), fs.ReadUInt(), fs.ReadUInt(),
      fs.ReadUInt() },
00402                     AchVendID = new byte[] { (byte)fs.ReadByte(), (byte)fs.ReadByte(),
      (byte)fs.ReadByte(), (byte)fs.ReadByte() },
00403                     FsSelection = fs.ReadUShort(),
00404                     FsFirstCharIndex = fs.ReadUShort(),
00405                     FsLastCharIndex = fs.ReadUShort()
00406                 };
00407
00408                 if (os2.Version > 0 || os2.Version == 0 && this.TableOffsets["OS/2"].Length > 68)
00409                 {
00410                     os2.STypoAscender = fs.ReadShort();
00411                     os2.STypoDescender = fs.ReadShort();
00412                     os2.STypoLineGap = fs.ReadShort();
00413                     os2.UsWinAscent = fs.ReadUShort();
00414                     os2.UsWinDescent = fs.ReadUShort();
00415
00416                     if (os2.Version >= 1)
00417                     {
00418                         os2.UlCodePageRange = new uint[] { fs.ReadUInt(), fs.ReadUInt() };
```

```
00419
00420                           if (os2.Version >= 2)
00421                           {
00422                               os2.SxHeight = fs.ReadShort();
00423                               os2.SCapHeight = fs.ReadShort();
00424                               os2.UsDefaultChar = fs.ReadUShort();
00425                               os2.UsBreakChar = fs.ReadUShort();
00426                               os2.UsMaxContext = fs.ReadUShort();
00427
00428                               if (os2.Version >= 5)
00429                               {
00430                                   os2.UsLowerOpticalPointSize = fs.ReadUShort();
00431                                   os2.UsUpperOpticalPointSize = fs.ReadUShort();
00432                               }
00433                           }
00434                       }
00435                   }
00436
00437
00438                   Tables.Add("OS/2", os2);
00439               }
00440
00441               if (this.TableOffsets.ContainsKey("post"))
00442               {
00443                   fs.Seek(this.TableOffsets["post"].Offset, SeekOrigin.Begin);
00444
00445                   TrueTypePostTable post = new TrueTypePostTable()
00446                   {
00447                       Version = fs.ReadFixed(),
00448                       ItalicAngle = fs.ReadFixed(),
00449                       UnderlinePosition = fs.ReadShort(),
00450                       UnderlineThickness = fs.ReadShort(),
00451                       IsFixedPitch = fs.ReadUInt(),
00452                       MinMemType42 = fs.ReadUInt(),
00453                       MaxMemType42 = fs.ReadUInt(),
00454                       MinMemType1 = fs.ReadUInt(),
00455                       MaxMemType1 = fs.ReadUInt()
00456                   };
00457
00458                   if (post.Version.Bits == 0x00020000)
00459                   {
00460                       post.NumGlyphs = fs.ReadUShort();
00461                       post.GlyphNameIndex = new ushort[post.NumGlyphs];
00462
00463                       int numberNewGlyphs = 0;
00464
00465                       for (int i = 0; i < post.NumGlyphs; i++)
00466                       {
00467                           post.GlyphNameIndex[i] = fs.ReadUShort();
00468                           if (post.GlyphNameIndex[i] >= 258)
00469                           {
00470                               numberNewGlyphs++;
00471                           }
00472                       }
00473
00474                       post.Names = new string[numberNewGlyphs];
00475
00476                       for (int i = 0; i < numberNewGlyphs; i++)
00477                       {
00478                           post.Names[i] = fs.ReadPascalString();
00479                       }
00480                   }
00481                   Tables.Add("post", post);
00482               }
00483
00484               if (this.TableOffsets.ContainsKey("name"))
00485               {
00486                   fs.Seek(this.TableOffsets["name"].Offset, SeekOrigin.Begin);
00487                   Tables.Add("name", new TrueTypeNameTable(this.TableOffsets["name"].Offset, fs));
00488               }
00489
00490               fs.Seek(this.TableOffsets["loca"].Offset, SeekOrigin.Begin);
00491               Tables.Add("loca", new TrueTypeLocaTable(fs,
      ((TrueTypeMaxpTable)this.Tables["maxp"]).NumGlyphs,
      ((TrueTypeHeadTable)this.Tables["head"]).IndexToLocFormat == 0));
00492
00493               if (this.TableOffsets.ContainsKey("cvt "))
00494               {
00495                   fs.Seek(this.TableOffsets["cvt "].Offset, SeekOrigin.Begin);
00496                   Tables.Add("cvt ", new TrueTypeRawTable(fs, this.TableOffsets["cvt "].Length));
00497               }
00498
00499               if (this.TableOffsets.ContainsKey("prep"))
00500               {
00501                   fs.Seek(this.TableOffsets["prep"].Offset, SeekOrigin.Begin);
00502                   Tables.Add("prep", new TrueTypeRawTable(fs, this.TableOffsets["prep"].Length));
00503               }
```

```
00504
00505             if (this.TableOffsets.ContainsKey("fpgm"))
00506             {
00507                 fs.Seek(this.TableOffsets["fpgm"].Offset, SeekOrigin.Begin);
00508                 Tables.Add("fpgm", new TrueTypeRawTable(fs, this.TableOffsets["fpgm"].Length));
00509             }
00510
00511             if (this.TableOffsets.ContainsKey("GPOS"))
00512             {
00513                 fs.Seek(this.TableOffsets["GPOS"].Offset, SeekOrigin.Begin);
00514                 Tables.Add("GPOS", new TrueTypeGPOSTable(fs));
00515             }
00516
00517             Glyph[] glyphs = new Glyph[((TrueTypeMaxpTable)this.Tables["maxp"]).NumGlyphs];
00518
00519             for (int i = 0; i < ((TrueTypeMaxpTable)this.Tables["maxp"]).NumGlyphs; i++)
00520             {
00521                 uint offset = ((TrueTypeLocaTable)Tables["loca"]).GetOffset(i);
00522                 uint length = ((TrueTypeLocaTable)Tables["loca"]).Lengths[i];
00523
00524                 if (length > 0)
00525                 {
00526                     fs.Seek(this.TableOffsets["glyf"].Offset + offset, SeekOrigin.Begin);
00527                     glyphs[i] = Glyph.Parse(fs);
00528                 }
00529                 else
00530                 {
00531                     glyphs[i] = new EmptyGlyph();
00532                 }
00533
00534             }
00535
00536             Tables.Add("glyf", new TrueTypeGlyfTable() { Glyphs = glyphs });
00537
00538             BuildGlyphCache();
00539         }
00540
00541 /// <summary>
00542 /// Represets a  TrueType name.
00543 /// </summary>
00544         public struct TrueTypeName
00545         {
00546 /// <summary>
00547 /// Describes a kind of TrueType name.
00548 /// </summary>
00549             public enum NameIdentifier
00550             {
00551 /// <summary>
00552 /// A font family name.
00553 /// </summary>
00554                 FontFamily = 1,
00555
00556 /// <summary>
00557 /// A full name for a font.
00558 /// </summary>
00559                 FullName = 4,
00560
00561 /// <summary>
00562 /// A PostScript name for a font.
00563 /// </summary>
00564                 PostScriptName = 6,
00565
00566 /// <summary>
00567 /// The preferred font family name for the font.
00568 /// </summary>
00569                 PreferredFamily = 16
00570             }
00571
00572 /// <summary>
00573 /// The kind of name represented by this struct.
00574 /// </summary>
00575             public NameIdentifier NameId;
00576
00577 /// <summary>
00578 /// The name represented by this struct.
00579 /// </summary>
00580             public string Name;
00581
00582             internal TrueTypeName(NameIdentifier identifier, string name)
00583             {
00584                 this.NameId = identifier;
00585                 this.Name = name;
00586             }
00587         }
00588
00589 /// <summary>
00590 /// Determines whether the file is a valid TrueType file and retrieves the list of names defined in
```

```
    it.
00591 /// </summary>
00592 /// <param name="file">The TrueType file.</param>
00593 /// <param name="names">The list of names that will be returned.  These will only include names with
    TrueType identifiers 1, 4, 6, or 16.</param>
00594 /// <returns>If an error occurred while opening the file, or the file does not contain a valid
    TrueType file, <see langword="false"/>.  If the TrueType file appears valid but does not contain a
    name table,
00595 /// <see langword="true"/>, but <paramref name="names"/> will be <see langword="null"/>.  Otherwise,
    <see langword="true"/> and <paramref name="names"/> will not be null
00596 /// (but it might still be empty).</returns>
00597         public static bool GetNames(string file, out List<TrueTypeName> names)
00598         {
00599             try
00600             {
00601                 using (FileStream fs = File.Open(file, FileMode.Open, FileAccess.Read,
    FileShare.Read))
00602                 {
00603                     return GetNames(fs, out names);
00604                 }
00605             }
00606             catch
00607             {
00608                 names = null;
00609                 return false;
00610             }
00611         }
00612 
00613 /// <summary>
00614 /// Determines whether the stream contains a valid TrueType file and retrieves the list of names
    defined in it.
00615 /// </summary>
00616 /// <param name="fs">The stream containing the TrueType file.</param>
00617 /// <param name="names">The list of names that will be returned.  These will only include names with
    TrueType identifiers 1, 4, 6, or 16.</param>
00618 /// <returns>If the stream does not contain a valid TrueType file, <see langword="false"/>.  If the
    TrueType file appears valid but does not contain a name table,
00619 /// <see langword="true"/>, but <paramref name="names"/> will be <see langword="null"/>.  Otherwise,
    <see langword="true"/> and <paramref name="names"/> will not be null
00620 /// (but it might still be empty).</returns>
00621         public static bool GetNames(Stream fs, out List<TrueTypeName> names)
00622         {
00623             try
00624             {
00625                 uint ScalarType = fs.ReadUInt();
00626 
00627                 if (ScalarType != 0x00010000 && ScalarType != 0x74727565)
00628                 {
00629                     names = null;
00630                     return false;
00631                 }
00632 
00633                 int NumTables = fs.ReadUShort();
00634                 fs.ReadUShort();
00635                 fs.ReadUShort();
00636                 fs.ReadUShort();
00637 
00638                 TrueTypeNameTable name = null;
00639 
00640                 for (int i = 0; i < NumTables; i++)
00641                 {
00642                     string key = fs.ReadString(4);
00643                     TrueTypeTableOffset offset = new TrueTypeTableOffset(fs.ReadUInt(), fs.ReadUInt(),
    fs.ReadUInt());
00644 
00645                     if (key == "name")
00646                     {
00647                         fs.Seek(offset.Offset, SeekOrigin.Begin);
00648                         name = new TrueTypeNameTable(offset.Offset, fs);
00649                         break;
00650                     }
00651                 }
00652 
00653                 names = null;
00654 
00655                 if (name != null)
00656                 {
00657                     names = new List<TrueTypeName>();
00658 
00659                     for (int i = 0; i < name.NameRecords.Length; i++)
00660                     {
00661                         if (name.NameRecords[i].NameID == 1 || name.NameRecords[i].NameID == 4 ||
    name.NameRecords[i].NameID == 6 || name.NameRecords[i].NameID == 16)
00662                         {
00663                             names.Add(new
    TrueTypeName((TrueTypeName.NameIdentifier)name.NameRecords[i].NameID, name.Name[i]));
00664                         }
```

```
00665                     }
00666                 }
00667 
00668                 return true;
00669             }
00670             catch
00671             {
00672                 names = null;
00673                 return false;
00674             }
00675         }
00676 
00677         private double[] GlyphWidthsCache = new double[256];
00678         private VerticalMetrics[] VerticalMetricsCache = new VerticalMetrics[256];
00679         private Bearings[] BearingsCache = new Bearings[256];
00680         private void BuildGlyphCache()
00681         {
00682             for (int i = 0; i < 256; i++)
00683             {
00684                 GlyphWidthsCache[i] = this.Get1000EmGlyphWidth(GetGlyphIndex((char)i));
00685                 VerticalMetricsCache[i] = this.Get1000EmGlyphVerticalMetrics(GetGlyphIndex((char)i));
00686                 BearingsCache[i] = this.Get1000EmGlyphBearings(GetGlyphIndex((char)i));
00687             }
00688         }
00689 
00690 /// <summary>
00691 /// Create a subset of the TrueType file, containing only the glyphs for the specified characters.
00692 /// </summary>
00693 /// <param name="charactersToInclude">A string containing the characters for which the glyphs should
      be included.</param>
00694 /// <param name="consolidateAt32">If true, the character map is rearranged so that the included glyphs
      start at the unicode U+0032 control point.</param>
00695 /// <param name="outputEncoding">If <paramref name="consolidateAt32"/> is true, entries will be added
      to this dictionary mapping the original characters to the new map (that starts at U+0033).</param>
00696 /// <returns>The subsetted TrueType file.</returns>
00697         [Pure]
00698         public TrueTypeFile SubsetFont(string charactersToInclude, bool consolidateAt32 = false,
      Dictionary<char, char> outputEncoding = null)
00699         {
00700             if (!this.HasCmap4Table())
00701             {
00702                 return this;
00703             }
00704             else
00705             {
00706 
00707                 TrueTypeHeadTable head = (TrueTypeHeadTable)this.Tables["head"];
00708 
00709                 TrueTypeHHeaTable originalHhea = (TrueTypeHHeaTable)this.Tables["hhea"];
00710                 TrueTypeMaxpTable originalMaxp = (TrueTypeMaxpTable)this.Tables["maxp"];
00711                 TrueTypeCmapTable originalCmap = (TrueTypeCmapTable)this.Tables["cmap"];
00712                 TrueTypeLocaTable originalLoca = (TrueTypeLocaTable)this.Tables["loca"];
00713                 TrueTypeGlyfTable originalGlyf = (TrueTypeGlyfTable)this.Tables["glyf"];
00714 
00715                 CmapTable4 cmap4 = null;
00716                 int platformId = -1;
00717                 int platformSpecificID = -1;
00718 
00719                 for (int i = 0; i < originalCmap.ActualCmapTables.Length; i++)
00720                 {
00721                     if (originalCmap.ActualCmapTables[i] != null &&
      originalCmap.ActualCmapTables[i].Format == 4 && originalCmap.SubTables[i].PlatformID == 3 &&
      originalCmap.SubTables[i].PlatformSpecificID == 1)
00722                     {
00723                         cmap4 = (CmapTable4)originalCmap.ActualCmapTables[i];
00724                         platformId = originalCmap.SubTables[i].PlatformID;
00725                         platformSpecificID = originalCmap.SubTables[i].PlatformSpecificID;
00726                         break;
00727                     }
00728                 }
00729 
00730                 if (cmap4 == null)
00731                 {
00732                     return this;
00733                 }
00734 
00735                 List<int> characterCodes = new List<int>();
00736 
00737                 for (int i = 0; i < charactersToInclude.Length; i++)
00738                 {
00739                     characterCodes.Add((int)charactersToInclude[i]);
00740                 }
00741 
00742                 characterCodes.Sort();
00743 
00744                 List<(int, int)> segments = new List<(int, int)>();
00745 
```

```
00746                 for (int i = 0; i < characterCodes.Count; i++)
00747                 {
00748                     if (segments.Count > 0 && segments.Last().Item1 + segments.Last().Item2 ==
      characterCodes[i])
00749                     {
00750                         segments[segments.Count - 1] = (segments[segments.Count - 1].Item1,
      segments[segments.Count - 1].Item2 + 1);
00751                     }
00752                     else
00753                     {
00754                         segments.Add((characterCodes[i], 1));
00755                     }
00756                 }
00757 
00758                 segments.Add((cmap4.StartCode.Last(), cmap4.EndCode.Last() - cmap4.StartCode.Last() +
      1));
00759 
00760                 int totalGlyphIndexCount = characterCodes.Count + 1;
00761 
00762                 segments.Sort((a, b) => a.Item1 + a.Item2 - b.Item1 - b.Item2);
00763 
00764                 ushort entrySelector = (ushort)Math.Floor(Math.Log(segments.Count, 2));
00765 
00766 
00767                 CmapTable4 newCmap4 = new CmapTable4()
00768                 {
00769                     Format = 4,
00770                     Length = (ushort)(16 + 8 * segments.Count),
00771                     Language = cmap4.Language,
00772                     SegCountX2 = (ushort)(2 * segments.Count),
00773                     SearchRange = (ushort)(2 * (1 << entrySelector)),
00774                     EntrySelector = entrySelector,
00775                     RangeShift = (ushort)(2 * segments.Count - (2 * (1 << entrySelector))),
00776                     EndCode = new ushort[segments.Count],
00777                     ReservedPad = cmap4.ReservedPad,
00778                     StartCode = new ushort[segments.Count],
00779                     IdDelta = new ushort[segments.Count],
00780                     IdRangeOffset = new ushort[segments.Count],
00781                     GlyphIndexArray = new ushort[0]
00782                 };
00783 
00784                 List<char> includedGlyphs = new List<char>
00785                 {
00786                     (char)0
00787                 };
00788 
00789                 List<ushort> glyphIndexArray = new List<ushort>();
00790 
00791                 if (!consolidateAt32)
00792                 {
00793                     for (int i = 0; i < segments.Count; i++)
00794                     {
00795                         newCmap4.EndCode[i] = (ushort)(segments[i].Item1 + segments[i].Item2 - 1);
00796                         newCmap4.StartCode[i] = (ushort)segments[i].Item1;
00797 
00798                         newCmap4.IdRangeOffset[i] = 0;
00799 
00800                         if (i < segments.Count - 1)
00801                         {
00802                             newCmap4.IdDelta[i] = (ushort)(includedGlyphs.Count -
      newCmap4.StartCode[i]);
00803 
00804                             for (int j = newCmap4.StartCode[i]; j <= newCmap4.EndCode[i]; j++)
00805                             {
00806                                 includedGlyphs.Add((char)j);
00807                             }
00808                         }
00809                         else
00810                         {
00811                             newCmap4.IdDelta[i] = 1;
00812                             newCmap4.IdRangeOffset[i] = 0;
00813                         }
00814                     }
00815                 }
00816                 else
00817                 {
00818                     for (int i = 0; i < segments.Count - 1; i++)
00819                     {
00820                         newCmap4.EndCode[i] = (ushort)(32 + includedGlyphs.Count + segments[i].Item2 -
      1);
00821                         newCmap4.StartCode[i] = (ushort)(32 + includedGlyphs.Count);
00822                         newCmap4.IdRangeOffset[i] = 0;
00823 
00824                         if (i < segments.Count - 1)
00825                         {
00826                             newCmap4.IdDelta[i] = (ushort)(includedGlyphs.Count -
      newCmap4.StartCode[i]);
```

```
00827
00828                               for (int j = segments[i].Item1; j < segments[i].Item1 + segments[i].Item2;
      j++)
00829                               {
00830                                   includedGlyphs.Add((char)j);
00831                               }
00832                           }
00833                           else
00834                           {
00835                               newCmap4.IdDelta[i] = 1;
00836                               newCmap4.IdRangeOffset[i] = 0;
00837                           }
00838                       }
00839
00840                       {
00841                           int i = segments.Count - 1;
00842                           newCmap4.EndCode[i] = (ushort)(segments[i].Item1 + segments[i].Item2 - 1);
00843                           newCmap4.StartCode[i] = (ushort)segments[i].Item1;
00844                           newCmap4.IdRangeOffset[i] = 0;
00845
00846                           if (i < segments.Count - 1)
00847                           {
00848                               newCmap4.IdDelta[i] = (ushort)(includedGlyphs.Count -
      newCmap4.StartCode[i]);
00849
00850                               for (int j = newCmap4.StartCode[i]; j <= newCmap4.EndCode[i]; j++)
00851                               {
00852                                   includedGlyphs.Add((char)j);
00853                               }
00854                           }
00855                           else
00856                           {
00857                               newCmap4.IdDelta[i] = 1;
00858                               newCmap4.IdRangeOffset[i] = 0;
00859                           }
00860                       }
00861
00862                       if (outputEncoding != null)
00863                       {
00864                           for (int i = 1; i < includedGlyphs.Count; i++)
00865                           {
00866                               outputEncoding.Add(includedGlyphs[i], (char)(32 + i));
00867                           }
00868                       }
00869                   }
00870
00871
00872                   newCmap4.GlyphIndexArray = glyphIndexArray.ToArray();
00873                   newCmap4.Length += (ushort)(newCmap4.GlyphIndexArray.Length * 2);
00874
00875                   TrueTypeCmapTable cmap = new TrueTypeCmapTable() { ActualCmapTables = new ICmapTable[]
      { newCmap4 }, NumberSubTables = 1, Version = 0, SubTables = new CmapSubTable[] { new
      CmapSubTable((ushort)platformId, (ushort)platformSpecificID, 12) } };
00876
00877
00878                   List<Glyph> glyphs = new List<Glyph>();
00879                   List<int> originalGlyphIndices = new List<int>();
00880
00881                   for (int i = 0; i < includedGlyphs.Count; i++)
00882                   {
00883                       int index = this.GetGlyphIndex(includedGlyphs[i]);
00884                       originalGlyphIndices.Add(index);
00885                       glyphs.Add(originalGlyf.Glyphs[index].Clone());
00886                   }
00887
00888                   for (int i = 0; i < glyphs.Count; i++)
00889                   {
00890                       if (glyphs[i] is CompositeGlyph)
00891                       {
00892                           for (int j = 0; j < ((CompositeGlyph)glyphs[i]).GlyphIndex.Length; j++)
00893                           {
00894                               if
      (originalGlyphIndices.Contains(((CompositeGlyph)glyphs[i]).GlyphIndex[j]))
00895                               {
00896                                   ((CompositeGlyph)glyphs[i]).GlyphIndex[j] =
      (ushort)originalGlyphIndices.IndexOf(((CompositeGlyph)glyphs[i]).GlyphIndex[j]);
00897                               }
00898                               else
00899                               {
00900                                   originalGlyphIndices.Add(((CompositeGlyph)glyphs[i]).GlyphIndex[j]);
00901
      glyphs.Add(originalGlyf.Glyphs[((CompositeGlyph)glyphs[i]).GlyphIndex[j]]);
00902                                   ((CompositeGlyph)glyphs[i]).GlyphIndex[j] =
      (ushort)originalGlyphIndices.IndexOf(((CompositeGlyph)glyphs[i]).GlyphIndex[j]);
00903                               }
00904                           }
00905                       }
```

```
00906                 }
00907
00908                 TrueTypeGlyfTable glyf = new TrueTypeGlyfTable() { Glyphs = glyphs.ToArray() };
00909
00910                 TrueTypeLocaTable loca = new TrueTypeLocaTable(glyphs.Count, head.IndexToLocFormat ==
      0);
00911
00912                 for (int i = 0; i < glyphs.Count + 1; i++)
00913                 {
00914                     if (i == 0)
00915                     {
00916                         loca.SetOffset(0, 0);
00917                         loca.Lengths[i] = (uint)(glyphs[i].GetBytes()).Length;
00918                     }
00919                     else
00920                     {
00921                         if (i < glyphs.Count)
00922                         {
00923                             loca.Lengths[i] = (uint)(glyphs[i].GetBytes()).Length;
00924                         }
00925
00926                         loca.SetOffset(i, loca.GetOffset(i - 1) + loca.Lengths[i - 1]);
00927                     }
00928                 }
00929
00930                 TrueTypeHHeaTable hhea = new TrueTypeHHeaTable()
00931                 {
00932                     Version = originalHhea.Version,
00933                     Ascent = originalHhea.Ascent,
00934                     Descent = originalHhea.Descent,
00935                     LineGap = originalHhea.LineGap,
00936                     AdvanceWidthMax = originalHhea.AdvanceWidthMax,
00937                     MinLeftSideBearing = originalHhea.MinLeftSideBearing,
00938                     MinRightSideBearing = originalHhea.MinRightSideBearing,
00939                     XMaxExtent = originalHhea.XMaxExtent,
00940                     CaretSlopeRise = originalHhea.CaretSlopeRise,
00941                     CaretSlopeRun = originalHhea.CaretSlopeRun,
00942                     CaretOffset = originalHhea.CaretOffset,
00943                     MetricDataFormat = originalHhea.MetricDataFormat,
00944                     NumOfLongHorMetrics = (ushort)glyphs.Count
00945                 };
00946
00947
00948                 LongHorFixed[] metrics = new LongHorFixed[glyphs.Count];
00949
00950                 for (int i = 0; i < glyphs.Count; i++)
00951                 {
00952                     metrics[i] = this.GetGlyphMetrics(originalGlyphIndices[i]);
00953                 }
00954
00955                 TrueTypeHmtxTable hmtx = new TrueTypeHmtxTable()
00956                 {
00957                     LeftSideBearing = new short[0],
00958                     HMetrics = metrics
00959                 };
00960
00961                 TrueTypeMaxpTable maxp = new TrueTypeMaxpTable()
00962                 {
00963                     Version = originalMaxp.Version,
00964                     NumGlyphs = (ushort)glyphs.Count,
00965                     MaxPoints = originalMaxp.MaxPoints,
00966                     MaxContours = originalMaxp.MaxContours,
00967                     MaxComponentPoints = originalMaxp.MaxComponentPoints,
00968                     MaxComponentContours = originalMaxp.MaxComponentContours,
00969                     MaxZones = originalMaxp.MaxZones,
00970                     MaxTwilightPoints = originalMaxp.MaxTwilightPoints,
00971                     MaxStorage = originalMaxp.MaxStorage,
00972                     MaxFunctionDefs = originalMaxp.MaxFunctionDefs,
00973                     MaxInstructionDefs = originalMaxp.MaxInstructionDefs,
00974                     MaxStackElements = originalMaxp.MaxStackElements,
00975                     MaxSizeOfInstructions = originalMaxp.MaxSizeOfInstructions,
00976                     MaxComponentElements = originalMaxp.MaxComponentElements,
00977                     MaxComponentDepth = originalMaxp.MaxComponentDepth
00978                 };
00979
00980                 Dictionary<string, ITrueTypeTable> newTables = new Dictionary<string,
      ITrueTypeTable>() {
00981                     {"head", head },
00982                     {"hhea", hhea },
00983                     {"maxp", maxp },
00984                 };
00985
00986                 if (Tables.ContainsKey("OS/2"))
00987                 {
00988                     TrueTypeOS2Table originalOS2 = (TrueTypeOS2Table)Tables["OS/2"];
00989
00990                     TrueTypeOS2Table os2 = new TrueTypeOS2Table()
```

```
00991                   {
00992                       AchVendID = originalOS2.AchVendID,
00993                       FsSelection = originalOS2.FsSelection,
00994                       FsType = originalOS2.FsType,
00995                       Panose = originalOS2.Panose,
00996                       SCapHeight = originalOS2.SCapHeight,
00997                       SFamilyClass = originalOS2.SFamilyClass,
00998                       SFamilySubClass = originalOS2.SFamilySubClass,
00999                       STypoAscender = originalOS2.STypoAscender,
01000                       STypoDescender = originalOS2.STypoDescender,
01001                       STypoLineGap = originalOS2.STypoLineGap,
01002                       SxHeight = originalOS2.SxHeight,
01003                       UlCodePageRange = originalOS2.UlCodePageRange,
01004                       UlUnicodeRange = originalOS2.UlUnicodeRange,
01005                       UsLowerOpticalPointSize = originalOS2.UsLowerOpticalPointSize,
01006                       UsBreakChar = originalOS2.UsBreakChar,
01007                       UsDefaultChar = originalOS2.UsDefaultChar,
01008                       UsMaxContext = originalOS2.UsMaxContext,
01009                       UsUpperOpticalPointSize = originalOS2.UsUpperOpticalPointSize,
01010                       UsWeightClass = originalOS2.UsWeightClass,
01011                       UsWidthClass = originalOS2.UsWidthClass,
01012                       UsWinAscent = originalOS2.UsWinAscent,
01013                       UsWinDescent = originalOS2.UsWinDescent,
01014                       Version = originalOS2.Version,
01015                       XAvgCharWidth = originalOS2.XAvgCharWidth,
01016                       YStrikeoutPosition = originalOS2.YStrikeoutPosition,
01017                       YStrikeoutSize = originalOS2.YStrikeoutSize,
01018                       YSubscriptXOffset = originalOS2.YSubscriptXOffset,
01019                       YSubscriptXSize = originalOS2.YSubscriptXSize,
01020                       YSubscriptYOffset = originalOS2.YSubscriptYOffset,
01021                       YSubscriptYSize = originalOS2.YSubscriptYSize,
01022                       YSuperscriptXOffset = originalOS2.YSuperscriptXOffset,
01023                       YSuperscriptXSize = originalOS2.YSuperscriptXSize,
01024                       YSuperscriptYOffset = originalOS2.YSuperscriptYOffset,
01025                       YSuperscriptYSize = originalOS2.YSuperscriptYSize,
01026                       FsFirstCharIndex = (ushort)characterCodes[0],
01027                       FsLastCharIndex = (ushort)characterCodes[characterCodes.Count - 1]
01028                   };
01029
01030                   newTables.Add("OS/2", os2);
01031               }
01032
01033               newTables.Add("hmtx", hmtx);
01034               newTables.Add("cmap", cmap);
01035               if (Tables.ContainsKey("fpgm"))
01036               {
01037                   TrueTypeRawTable fpgm = (TrueTypeRawTable)Tables["fpgm"];
01038                   newTables.Add("fpgm", fpgm);
01039               }
01040               if (Tables.ContainsKey("prep"))
01041               {
01042                   TrueTypeRawTable prep = (TrueTypeRawTable)Tables["prep"];
01043                   newTables.Add("prep", prep);
01044               }
01045               if (Tables.ContainsKey("cvt "))
01046               {
01047                   TrueTypeRawTable cvt = (TrueTypeRawTable)Tables["cvt "];
01048                   newTables.Add("cvt ", cvt);
01049               }
01050               newTables.Add("loca", loca);
01051               newTables.Add("glyf", glyf);
01052
01053               if (Tables.ContainsKey("name"))
01054               {
01055                   TrueTypeNameTable name = (TrueTypeNameTable)Tables["name"];
01056                   newTables.Add("name", name);
01057               }
01058
01059               if (Tables.ContainsKey("post"))
01060               {
01061                   TrueTypePostTable oldPost = (TrueTypePostTable)Tables["post"];
01062
01063                   TrueTypePostTable post = new TrueTypePostTable()
01064                   {
01065                       //We don't really care about PostScript names
01066                       Version = new Fixed(0x00030000, 16),
01067                       ItalicAngle = oldPost.ItalicAngle,
01068                       UnderlinePosition = oldPost.UnderlinePosition,
01069                       UnderlineThickness = oldPost.UnderlineThickness,
01070                       IsFixedPitch = oldPost.IsFixedPitch,
01071                       MinMemType42 = oldPost.MinMemType42,
01072                       MaxMemType42 = oldPost.MaxMemType42,
01073                       MinMemType1 = oldPost.MinMemType1,
01074                       MaxMemType1 = oldPost.MaxMemType1
01075                   };
01076
01077                   newTables.Add("post", post);
```

```
01078                 }
01079
01080                 TrueTypeFile newFile = new TrueTypeFile(newTables);
01081
01082                 return newFile;
01083             }
01084
01085         }
01086
01087         internal class TrueTypeGlyfTable :  ITrueTypeTable
01088         {
01089             public Glyph[] Glyphs;
01090
01091             public byte[] GetBytes()
01092             {
01093                 using (MemoryStream ms = new MemoryStream())
01094                 {
01095                     for (int i = 0; i < Glyphs.Length; i++)
01096                     {
01097                         byte[] glyph = Glyphs[i].GetBytes();
01098                         ms.Write(glyph, 0, glyph.Length);
01099                     }
01100                     return ms.ToArray();
01101                 }
01102             }
01103
01104         }
01105
01106         internal abstract class Glyph
01107         {
01108             public short NumberOfContours { get; set; }
01109             public short XMin { get; set; }
01110             public short YMin { get; set; }
01111             public short XMax { get; set; }
01112             public short YMax { get; set; }
01113
01114             public abstract byte[] GetBytes();
01115             public abstract Glyph Clone();
01116             public static Glyph Parse(Stream sr)
01117             {
01118                 short numOfContours = sr.ReadShort();
01119                 if (numOfContours >= 0)
01120                 {
01121                     return new SimpleGlyph(sr, numOfContours);
01122                 }
01123                 else
01124                 {
01125                     return new CompositeGlyph(sr, numOfContours);
01126                 }
01127             }
01128
01129             public abstract TrueTypePoint[][] GetGlyphPath(double size, int emSize, Glyph[]
      glyphCollection);
01130         }
01131
01132         internal class EmptyGlyph :  Glyph
01133         {
01134             public override byte[] GetBytes()
01135             {
01136                 return new byte[0];
01137             }
01138
01139             public override Glyph Clone()
01140             {
01141                 return new EmptyGlyph();
01142             }
01143
01144             public override TrueTypePoint[][] GetGlyphPath(double size, int emSize, Glyph[]
      glyphCollection)
01145             {
01146                 return new TrueTypePoint[0][];
01147             }
01148         }
01149         internal class CompositeGlyph :  Glyph
01150         {
01151             public ushort[] Flags { get; set; }
01152             public ushort[] GlyphIndex { get; set; }
01153             public byte[][] Argument1 { get; set; }
01154             public byte[][] Argument2 { get; set; }
01155             public byte[][] TransformationOption { get; set; }
01156             public ushort NumInstructions { get; set; }
01157             public byte[] Instructions { get; set; }
01158
01159             private CompositeGlyph() { }
01160             public override Glyph Clone()
01161             {
01162                 CompositeGlyph tbr = new CompositeGlyph()
```

```
01163                 {
01164                     NumberOfContours = this.NumberOfContours,
01165                     XMin = this.XMin,
01166                     YMin = this.YMin,
01167                     XMax = this.XMax,
01168                     YMax = this.YMax,
01169 
01170                     Flags = new ushort[this.Flags.Length],
01171 
01172                     GlyphIndex = new ushort[this.GlyphIndex.Length],
01173                     Argument1 = new byte[this.Argument1.Length][],
01174                     Argument2 = new byte[this.Argument2.Length][],
01175                     TransformationOption = new byte[this.TransformationOption.Length][],
01176                     NumInstructions = this.NumInstructions
01177                 };
01178                 this.Flags.CopyTo(tbr.Flags, 0);
01179 
01180                 this.GlyphIndex.CopyTo(tbr.GlyphIndex, 0);
01181 
01182                 for (int i = 0; i < this.Argument1.Length; i++)
01183                 {
01184                     tbr.Argument1[i] = (byte[])this.Argument1[i].Clone();
01185                 }
01186 
01187 
01188                 for (int i = 0; i < this.Argument2.Length; i++)
01189                 {
01190                     tbr.Argument2[i] = (byte[])this.Argument2[i].Clone();
01191                 }
01192 
01193 
01194                 for (int i = 0; i < this.TransformationOption.Length; i++)
01195                 {
01196                     tbr.TransformationOption[i] = (byte[])this.TransformationOption[i].Clone();
01197                 }
01198 
01199 
01200 
01201                 if (this.Instructions != null)
01202                 {
01203                     tbr.Instructions = new byte[this.Instructions.Length];
01204                     this.Instructions.CopyTo(tbr.Instructions, 0);
01205                 }
01206                 else
01207                 {
01208                     tbr.Instructions = null;
01209                 }
01210 
01211                 return tbr;
01212             }
01213             public CompositeGlyph(Stream sr, short numberOfContours) :  base()
01214             {
01215                 this.NumberOfContours = numberOfContours;
01216                 this.XMin = sr.ReadShort();
01217                 this.YMin = sr.ReadShort();
01218                 this.XMax = sr.ReadShort();
01219                 this.YMax = sr.ReadShort();
01220 
01221                 List<ushort> flags = new List<ushort>();
01222                 List<ushort> glyphIndex = new List<ushort>();
01223                 List<byte[]> argument1 = new List<byte[]>();
01224                 List<byte[]> argument2 = new List<byte[]>();
01225                 List<byte[]> transformationOption = new List<byte[]>();
01226 
01227                 bool moreComponents = true;
01228 
01229                 while (moreComponents)
01230                 {
01231                     flags.Add(sr.ReadUShort());
01232 
01233                     moreComponents = (flags.Last() & 0x0020) != 0;
01234                     glyphIndex.Add(sr.ReadUShort());
01235 
01236                     if ((flags.Last() & 0x0001) != 0)
01237                     {
01238                         argument1.Add(new byte[] { (byte)sr.ReadByte(), (byte)sr.ReadByte() });
01239                         argument2.Add(new byte[] { (byte)sr.ReadByte(), (byte)sr.ReadByte() });
01240                     }
01241                     else
01242                     {
01243                         argument1.Add(new byte[] { (byte)sr.ReadByte() });
01244                         argument2.Add(new byte[] { (byte)sr.ReadByte() });
01245                     }
01246 
01247                     if ((flags.Last() & 0x0008) != 0)
01248                     {
01249                         transformationOption.Add(new byte[] { (byte)sr.ReadByte(), (byte)sr.ReadByte()
```

```
    });
01250                     }
01251                     else if ((flags.Last() & 0x0040) != 0)
01252                     {
01253                         transformationOption.Add(new byte[] { (byte)sr.ReadByte(),
    (byte)sr.ReadByte(), (byte)sr.ReadByte(), (byte)sr.ReadByte() });
01254                     }
01255                     else if ((flags.Last() & 0x0080) != 0)
01256                     {
01257                         transformationOption.Add(new byte[] { (byte)sr.ReadByte(),
    (byte)sr.ReadByte(), (byte)sr.ReadByte(), (byte)sr.ReadByte(), (byte)sr.ReadByte(),
    (byte)sr.ReadByte(), (byte)sr.ReadByte(), (byte)sr.ReadByte() });
01258                     }
01259                     else
01260                     {
01261                         transformationOption.Add(new byte[] { });
01262                     }
01263                 }
01264
01265                 this.Flags = flags.ToArray();
01266                 this.GlyphIndex = glyphIndex.ToArray();
01267                 this.Argument1 = argument1.ToArray();
01268                 this.Argument2 = argument2.ToArray();
01269                 this.TransformationOption = transformationOption.ToArray();
01270
01271                 if ((flags.Last() & 0x0100) != 0)
01272                 {
01273                     this.NumInstructions = sr.ReadUShort();
01274                     this.Instructions = new byte[this.NumInstructions];
01275                     sr.Read(this.Instructions, 0, this.NumInstructions);
01276                 }
01277                 else
01278                 {
01279                     this.NumInstructions = 0;
01280                     this.Instructions = null;
01281                 }
01282             }
01283
01284             public override byte[] GetBytes()
01285             {
01286                 using (MemoryStream ms = new MemoryStream())
01287                 {
01288                     ms.WriteShort(this.NumberOfContours);
01289                     ms.WriteShort(this.XMin);
01290                     ms.WriteShort(this.YMin);
01291                     ms.WriteShort(this.XMax);
01292                     ms.WriteShort(this.YMax);
01293
01294                     for (int i = 0; i < this.Flags.Length; i++)
01295                     {
01296                         ms.WriteUShort(this.Flags[i]);
01297                         ms.WriteUShort(this.GlyphIndex[i]);
01298                         ms.Write(this.Argument1[i], 0, this.Argument1[i].Length);
01299                         ms.Write(this.Argument2[i], 0, this.Argument2[i].Length);
01300                         ms.Write(this.TransformationOption[i], 0,
    this.TransformationOption[i].Length);
01301                     }
01302
01303                     if (this.NumInstructions > 0)
01304                     {
01305                         ms.WriteUShort(this.NumInstructions);
01306                         ms.Write(this.Instructions, 0, this.NumInstructions);
01307                     }
01308
01309                     if (ms.Length % 2 != 0)
01310                     {
01311                         ms.WriteByte(0);
01312                     }
01313
01314                     return ms.ToArray();
01315                 }
01316             }
01317
01318             public override TrueTypePoint[][] GetGlyphPath(double size, int emSize, Glyph[]
    glyphCollection)
01319             {
01320                 List<short> argument1 = new List<short>();
01321                 List<short> argument2 = new List<short>();
01322
01323                 for (int i = 0; i < this.Argument1.Length; i++)
01324                 {
01325                     if (this.Argument1[i].Length == 1)
01326                     {
01327                         argument1.Add(this.Argument1[i][0]);
01328                     }
01329                     else if (this.Argument1[i].Length == 2)
01330                     {
```

```
01331                           argument1.Add((short)((this.Argument1[i][0] « 8) + this.Argument1[i][1]));
01332                       }
01333                   }
01334
01335                   for (int i = 0; i < this.Argument2.Length; i++)
01336                   {
01337                       if (this.Argument2[i].Length == 1)
01338                       {
01339                           argument2.Add(this.Argument2[i][0]);
01340                       }
01341                       else if (this.Argument2[i].Length == 2)
01342                       {
01343                           argument2.Add((short)((this.Argument2[i][0] « 8) + this.Argument2[i][1]));
01344                       }
01345                   }
01346
01347                   List<double[]> transformationOption = new List<double[]>();
01348
01349                   for (int i = 0; i < this.TransformationOption.Length; i++)
01350                   {
01351                       if (this.TransformationOption[i].Length == 0)
01352                       {
01353                           transformationOption.Add(new double[] { });
01354                       }
01355                       else if (this.TransformationOption[i].Length == 2)
01356                       {
01357                           double val = (double)((this.TransformationOption[i][0] « 8) +
      this.TransformationOption[i][1]) / (1 « 14);
01358                           transformationOption.Add(new double[] { val });
01359                       }
01360                       else if (this.TransformationOption[i].Length == 4)
01361                       {
01362                           double val1 = (double)((this.TransformationOption[i][0] « 8) +
      this.TransformationOption[i][1]) / (1 « 14);
01363                           double val2 = (double)((this.TransformationOption[i][2] « 8) +
      this.TransformationOption[i][3]) / (1 « 14);
01364                           transformationOption.Add(new double[] { val1, val2 });
01365                       }
01366                       else if (this.TransformationOption[i].Length == 8)
01367                       {
01368                           double val1 = (double)((this.TransformationOption[i][0] « 8) +
      this.TransformationOption[i][1]) / (1 « 14);
01369                           double val2 = (double)((this.TransformationOption[i][2] « 8) +
      this.TransformationOption[i][3]) / (1 « 14);
01370                           double val3 = (double)((this.TransformationOption[i][4] « 8) +
      this.TransformationOption[i][5]) / (1 « 14);
01371                           double val4 = (double)((this.TransformationOption[i][6] « 8) +
      this.TransformationOption[i][7]) / (1 « 14);
01372                           transformationOption.Add(new double[] { val1, val2, val3, val4 });
01373                       }
01374                   }
01375
01376                   List<TrueTypePoint[]> tbr = new List<TrueTypePoint[]>();
01377
01378
01379                   for (int i = 0; i < this.GlyphIndex.Length; i++)
01380                   {
01381                       TrueTypePoint[][] componentContours =
      glyphCollection[this.GlyphIndex[i]].GetGlyphPath(size, emSize, glyphCollection);
01382
01383                       double[,] transformMatrix = new double[,] { { 1, 0 }, { 0, 1 } };
01384
01385                       if ((Flags[i] & 0x0008) != 0)
01386                       {
01387                           transformMatrix[0, 0] = transformationOption[i][0];
01388                           transformMatrix[1, 1] = transformationOption[i][0];
01389                       }
01390                       else if ((Flags[i] & 0x0040) != 0)
01391                       {
01392                           transformMatrix[0, 0] = transformationOption[i][0];
01393                           transformMatrix[1, 1] = transformationOption[i][1];
01394                       }
01395                       else if ((Flags[i] & 0x0080) != 0)
01396                       {
01397                           transformMatrix[0, 0] = transformationOption[i][0];
01398                           transformMatrix[0, 1] = transformationOption[i][1];
01399                           transformMatrix[1, 0] = transformationOption[i][2];
01400                           transformMatrix[1, 1] = transformationOption[i][3];
01401                       }
01402
01403                       double deltaX = 0;
01404                       double deltaY = 0;
01405
01406                       if ((Flags[i] & 0x0002) != 0)
01407                       {
01408                           deltaX = argument1[i] * size / emSize;
01409                           deltaY = argument2[i] * size / emSize;
```

```
01410
01411                           if ((Flags[i] & 0x0800) != 0 && (Flags[i] & 0x1000) == 0)
01412                           {
01413                               deltaX *= Magnitude(Multiply(transformMatrix, new double[] { 1, 0 }));
01414                               deltaY *= Magnitude(Multiply(transformMatrix, new double[] { 0, 1 }));
01415                           }
01416                       }
01417                       else
01418                       {
01419                           TrueTypePoint reference = GetNthElementWhere(tbr, argument1[i], el =>
      el.IsDefinedPoint);
01420                           TrueTypePoint destination = GetNthElementWhere(componentContours,
      argument2[i], el => el.IsDefinedPoint);
01421
01422                           deltaX = reference.X - destination.X;
01423                           deltaY = reference.Y - destination.Y;
01424
01425                           //Note:  this would be the sensible behaviour (i.e.  make sure that reference
      and destination coincide after the transform), but apparently it is not the correct one.
01426                           /*double[] transfDestination = Multiply(transformMatrix, new double[] {
      destination.X, destination.Y });
01427
01428 deltaX = reference.X - transfDestination[0];
01429 deltaY = reference.Y - transfDestination[1];*/
01430                       }
01431
01432                       for (int j = 0; j < componentContours.Length; j++)
01433                       {
01434                           for (int k = 0; k < componentContours[j].Length; k++)
01435                           {
01436                               double[] transformed = Multiply(transformMatrix, new double[] {
      componentContours[j][k].X, componentContours[j][k].Y });
01437
01438                               componentContours[j][k] = new TrueTypePoint(transformed[0] + deltaX,
      transformed[1] + deltaY, componentContours[j][k].IsOnCurve, componentContours[j][k].IsDefinedPoint);
01439                           }
01440                       }
01441
01442                       tbr.AddRange(componentContours);
01443                   }
01444
01445                   return tbr.ToArray();
01446               }
01447           }
01448
01449           private static double Magnitude(double[] vector)
01450           {
01451               double tbr = 0;
01452
01453               for (int i = 0; i < vector.Length; i++)
01454               {
01455                   tbr += vector[i] * vector[i];
01456               }
01457
01458               return Math.Sqrt(tbr);
01459           }
01460
01461           private static double[] Multiply(double[,] matrix, double[] vector)
01462           {
01463               double[] tbr = new double[2];
01464
01465               tbr[0] = matrix[0, 0] * vector[0] + matrix[0, 1] * vector[1];
01466               tbr[1] = matrix[1, 0] * vector[0] + matrix[1, 1] * vector[1];
01467
01468               return tbr;
01469           }
01470
01471           private static T GetNthElementWhere<T>(IEnumerable<IEnumerable<T>> array, int n, Func<T, bool>
      condition)
01472           {
01473               int index = 0;
01474
01475               foreach (IEnumerable<T> arr1 in array)
01476               {
01477                   foreach (T el in arr1)
01478                   {
01479                       if (condition(el))
01480                       {
01481                           if (index == n)
01482                           {
01483                               return el;
01484                           }
01485                           index++;
01486                       }
01487                   }
01488               }
01489
```

```
01490             throw new IndexOutOfRangeException();
01491         }
01492 
01493 /// <summary>
01494 /// Represents a point in a TrueType path description.
01495 /// </summary>
01496         public struct TrueTypePoint
01497         {
01498 /// <summary>
01499 /// The horizontal coordinate of the point.
01500 /// </summary>
01501             public double X;
01502 
01503 /// <summary>
01504 /// The vertical coordinate of the point.
01505 /// </summary>
01506             public double Y;
01507 
01508 /// <summary>
01509 /// Whether the point is a point on the curve, or a control point of a quadratic Bezier curve.
01510 /// </summary>
01511             public bool IsOnCurve;
01512 
01513             internal bool IsDefinedPoint;
01514 
01515             internal TrueTypePoint(double x, double y, bool onCurve, bool isDefinedPoint)
01516             {
01517                 this.X = x;
01518                 this.Y = y;
01519                 this.IsOnCurve = onCurve;
01520                 this.IsDefinedPoint = isDefinedPoint;
01521             }
01522         }
01523 
01524         internal class SimpleGlyph :  Glyph
01525         {
01526             public ushort[] EndPtsOfContours { get; set; }
01527             public ushort InstructionLength { get; set; }
01528             public byte[] Instructions { get; set; }
01529             public byte[] Flags { get; set; }
01530             public byte[] XCoordinates { get; set; }
01531             public byte[] YCoordinates { get; set; }
01532 
01533             private SimpleGlyph() { }
01534             public override Glyph Clone()
01535             {
01536                 SimpleGlyph tbr = new SimpleGlyph
01537                 {
01538                     NumberOfContours = this.NumberOfContours,
01539                     XMin = this.XMin,
01540                     YMin = this.YMin,
01541                     XMax = this.XMax,
01542                     YMax = this.YMax,
01543 
01544                     EndPtsOfContours = new ushort[this.EndPtsOfContours.Length],
01545                     InstructionLength = this.InstructionLength,
01546                     Instructions = new byte[this.Instructions.Length],
01547                     Flags = new byte[this.Flags.Length],
01548                     XCoordinates = new byte[this.XCoordinates.Length],
01549                     YCoordinates = new byte[this.YCoordinates.Length]
01550                 };
01551 
01552                 this.EndPtsOfContours.CopyTo(tbr.EndPtsOfContours, 0);
01553                 this.Instructions.CopyTo(tbr.Instructions, 0);
01554                 this.Flags.CopyTo(tbr.Flags, 0);
01555                 this.XCoordinates.CopyTo(tbr.XCoordinates, 0);
01556                 this.YCoordinates.CopyTo(tbr.YCoordinates, 0);
01557 
01558                 return tbr;
01559             }
01560 
01561             public SimpleGlyph(Stream sr, short numberOfContours) :  base()
01562             {
01563                 this.NumberOfContours = numberOfContours;
01564                 this.XMin = sr.ReadShort();
01565                 this.YMin = sr.ReadShort();
01566                 this.XMax = sr.ReadShort();
01567                 this.YMax = sr.ReadShort();
01568 
01569                 this.EndPtsOfContours = new ushort[this.NumberOfContours];
01570                 for (int i = 0; i < this.NumberOfContours; i++)
01571                 {
01572                     this.EndPtsOfContours[i] = sr.ReadUShort();
01573                 }
01574 
01575                 this.InstructionLength = sr.ReadUShort();
01576                 this.Instructions = new byte[this.InstructionLength];
```

```
01577                 for (int i = 0; i < this.InstructionLength; i++)
01578                 {
01579                     this.Instructions[i] = (byte)sr.ReadByte();
01580                 }
01581 
01582                 List<byte> logicalFlags = new List<byte>();
01583                 List<byte> flags = new List<byte>();
01584 
01585                 int totalPoints = this.EndPtsOfContours[this.NumberOfContours - 1] + 1;
01586 
01587                 int countedPoints = 0;
01588 
01589                 while (countedPoints < totalPoints)
01590                 {
01591                     flags.Add((byte)sr.ReadByte());
01592                     logicalFlags.Add(flags.Last());
01593                     countedPoints++;
01594                     if ((flags.Last() & 0x08) != 0)
01595                     {
01596                         byte repeats = (byte)sr.ReadByte();
01597                         for (int i = 0; i < repeats; i++)
01598                         {
01599                             logicalFlags.Add(flags.Last());
01600                             countedPoints++;
01601                         }
01602                         flags.Add(repeats);
01603                     }
01604                 }
01605 
01606                 this.Flags = flags.ToArray();
01607 
01608                 List<byte> xCoordinates = new List<byte>();
01609 
01610                 for (int i = 0; i < totalPoints; i++)
01611                 {
01612                     bool isByte = (logicalFlags[i] & 0x02) != 0;
01613 
01614                     if (isByte)
01615                     {
01616                         xCoordinates.Add((byte)sr.ReadByte());
01617                     }
01618                     else if ((logicalFlags[i] & 0x10) == 0)
01619                     {
01620                         xCoordinates.Add((byte)sr.ReadByte());
01621                         xCoordinates.Add((byte)sr.ReadByte());
01622                     }
01623                 }
01624 
01625                 this.XCoordinates = xCoordinates.ToArray();
01626 
01627                 List<byte> yCoordinates = new List<byte>();
01628 
01629                 List<int> yCoordinateLengths = new List<int>();
01630 
01631                 for (int i = 0; i < totalPoints; i++)
01632                 {
01633                     bool isByte = (logicalFlags[i] & 0x04) != 0;
01634 
01635                     if (isByte)
01636                     {
01637                         yCoordinates.Add((byte)sr.ReadByte());
01638                         yCoordinateLengths.Add(1);
01639                     }
01640                     else if ((logicalFlags[i] & 0x20) == 0)
01641                     {
01642                         yCoordinates.Add((byte)sr.ReadByte());
01643                         yCoordinates.Add((byte)sr.ReadByte());
01644                         yCoordinateLengths.Add(2);
01645                     }
01646                     else
01647                     {
01648                         yCoordinateLengths.Add(0);
01649                     }
01650                 }
01651 
01652                 this.YCoordinates = yCoordinates.ToArray();
01653             }
01654 
01655             public override byte[] GetBytes()
01656             {
01657                 using (MemoryStream ms = new MemoryStream())
01658                 {
01659                     ms.WriteShort(this.NumberOfContours);
01660                     ms.WriteShort(this.XMin);
01661                     ms.WriteShort(this.YMin);
01662                     ms.WriteShort(this.XMax);
01663                     ms.WriteShort(this.YMax);
```

```
01664
01665                     for (int i = 0; i < this.EndPtsOfContours.Length; i++)
01666                     {
01667                         ms.WriteUShort(this.EndPtsOfContours[i]);
01668                     }
01669
01670                     ms.WriteUShort(this.InstructionLength);
01671                     ms.Write(this.Instructions, 0, this.InstructionLength);
01672                     ms.Write(this.Flags, 0, this.Flags.Length);
01673                     ms.Write(this.XCoordinates, 0, this.XCoordinates.Length);
01674                     ms.Write(this.YCoordinates, 0, this.YCoordinates.Length);
01675
01676                     if (ms.Length % 2 != 0)
01677                     {
01678                         ms.WriteByte(0);
01679                     }
01680
01681                     return ms.ToArray();
01682                 }
01683             }
01684
01685             public override TrueTypePoint[][] GetGlyphPath(double size, int emSize, Glyph[]
      glyphCollection)
01686             {
01687                 List<TrueTypePoint[]> contours = new List<TrueTypePoint[]>();
01688
01689                 List<TrueTypePoint> currentContour = new List<TrueTypePoint>();
01690
01691
01692                 List<byte> logicalFlags = new List<byte>();
01693
01694                 int totalPoints = this.EndPtsOfContours[this.NumberOfContours - 1] + 1;
01695
01696                 int countedPoints = 0;
01697
01698                 int index = 0;
01699
01700                 while (countedPoints < totalPoints)
01701                 {
01702                     logicalFlags.Add(this.Flags[index]);
01703                     index++;
01704                     countedPoints++;
01705                     if ((logicalFlags.Last() & 0x08) != 0)
01706                     {
01707                         byte repeats = this.Flags[index];
01708                         index++;
01709                         for (int i = 0; i < repeats; i++)
01710                         {
01711                             logicalFlags.Add(logicalFlags.Last());
01712                             countedPoints++;
01713                         }
01714                     }
01715                 }
01716
01717                 List<short> xCoordinates = new List<short>();
01718
01719                 index = 0;
01720
01721                 for (int i = 0; i < totalPoints; i++)
01722                 {
01723                     bool isByte = (logicalFlags[i] & 0x02) != 0;
01724
01725                     if (isByte)
01726                     {
01727                         if ((logicalFlags[i] & 0x10) != 0)
01728                         {
01729                             xCoordinates.Add(this.XCoordinates[index]);
01730                         }
01731                         else
01732                         {
01733                             xCoordinates.Add((short)(-this.XCoordinates[index]));
01734                         }
01735
01736                         index++;
01737                     }
01738                     else if ((logicalFlags[i] & 0x10) == 0)
01739                     {
01740                         xCoordinates.Add((short)((this.XCoordinates[index] « 8) +
      this.XCoordinates[index + 1]));
01741                         index += 2;
01742                     }
01743                     else
01744                     {
01745                         xCoordinates.Add(0);
01746                     }
01747                 }
01748
```

```
01749                 List<short> yCoordinates = new List<short>();
01750
01751                 index = 0;
01752
01753                 for (int i = 0; i < totalPoints; i++)
01754                 {
01755                     bool isByte = (logicalFlags[i] & 0x04) != 0;
01756
01757                     if (isByte)
01758                     {
01759                         if ((logicalFlags[i] & 0x20) != 0)
01760                         {
01761                             yCoordinates.Add(this.YCoordinates[index]);
01762                         }
01763                         else
01764                         {
01765                             yCoordinates.Add((short)(-this.YCoordinates[index]));
01766                         }
01767                         index++;
01768                     }
01769                     else if ((logicalFlags[i] & 0x20) == 0)
01770                     {
01771                         yCoordinates.Add((short)((this.YCoordinates[index] << 8) +
      this.YCoordinates[index + 1]));
01772                         index += 2;
01773                     }
01774                     else
01775                     {
01776                         yCoordinates.Add(0);
01777                     }
01778                 }
01779
01780                 int[] previousPoint = new int[2] { 0, 0 };
01781
01782                 for (int i = 0; i < totalPoints; i++)
01783                 {
01784                     int absoluteX = xCoordinates[i] + previousPoint[0];
01785                     int absoluteY = yCoordinates[i] + previousPoint[1];
01786
01787                     previousPoint[0] = absoluteX;
01788                     previousPoint[1] = absoluteY;
01789
01790                     bool onCurve = (logicalFlags[i] & 0x01) != 0;
01791
01792                     if (onCurve)
01793                     {
01794                         currentContour.Add(new TrueTypePoint(size * absoluteX / emSize, size *
      absoluteY / emSize, onCurve, true));
01795                     }
01796                     else
01797                     {
01798                         if (currentContour.Count > 0)
01799                         {
01800                             if (currentContour.Last().IsOnCurve)
01801                             {
01802                                 currentContour.Add(new TrueTypePoint(size * absoluteX / emSize, size *
      absoluteY / emSize, onCurve, true));
01803                             }
01804                             else
01805                             {
01806                                 double newX = size * absoluteX / emSize;
01807                                 double newY = size * absoluteY / emSize;
01808
01809                                 currentContour.Add(new TrueTypePoint((newX + currentContour.Last().X)
      * 0.5, (newY + currentContour.Last().Y) * 0.5, true, false));
01810                                 currentContour.Add(new TrueTypePoint(newX, newY, onCurve, true));
01811                             }
01812                         }
01813                         else
01814                         {
01815                             currentContour.Add(new TrueTypePoint(size * absoluteX / emSize, size *
      absoluteY / emSize, onCurve, true));
01816                         }
01817                     }
01818
01819                     if (this.EndPtsOfContours.Contains((ushort)i))
01820                     {
01821                         if (!currentContour[0].IsOnCurve)
01822                         {
01823                             if (currentContour.Last().IsOnCurve)
01824                             {
01825                                 currentContour.Insert(0, new TrueTypePoint(currentContour.Last().X,
      currentContour.Last().Y, currentContour.Last().IsOnCurve, false));
01826                             }
01827                             else
01828                             {
01829                                 currentContour.Insert(0, new TrueTypePoint((currentContour[0].X +
```

```
      currentContour.Last().X) * 0.5, (currentContour[0].Y + currentContour.Last().Y) * 0.5, true, false));
01830                             }
01831                         }
01832 
01833                         if (!currentContour.Last().IsOnCurve)
01834                         {
01835                             currentContour.Add(new TrueTypePoint(currentContour[0].X,
      currentContour[0].Y, currentContour[0].IsOnCurve, false));
01836                         }
01837 
01838                         contours.Add(currentContour.ToArray());
01839                         currentContour = new List<TrueTypePoint>();
01840                     }
01841                 }
01842 
01843                 return contours.ToArray();
01844             }
01845         }
01846 
01847         internal class TrueTypeLocaTable :  ITrueTypeTable
01848         {
01849             public ushort[] ShortOffsets { get; }
01850             public uint[] IntOffsets { get; }
01851             public uint[] Lengths { get; }
01852 
01853             public uint GetOffset(int index)
01854             {
01855                 if (IntOffsets == null)
01856                 {
01857                     return (uint)ShortOffsets[index] * 2;
01858                 }
01859                 else
01860                 {
01861                     return IntOffsets[index];
01862                 }
01863             }
01864 
01865             public void SetOffset(int index, uint value)
01866             {
01867                 if (IntOffsets == null)
01868                 {
01869                     ShortOffsets[index] = (ushort)(value / 2);
01870                 }
01871                 else
01872                 {
01873                     IntOffsets[index] = value;
01874                 }
01875             }
01876 
01877             public TrueTypeLocaTable(int numGlyphs, bool isShort)
01878             {
01879                 this.Lengths = new uint[numGlyphs];
01880 
01881                 if (isShort)
01882                 {
01883                     this.ShortOffsets = new ushort[numGlyphs + 1];
01884                 }
01885                 else
01886                 {
01887                     this.IntOffsets = new uint[numGlyphs + 1];
01888                 }
01889             }
01890 
01891             public TrueTypeLocaTable(Stream sr, int numGlyphs, bool isShort)
01892             {
01893                 this.Lengths = new uint[numGlyphs];
01894 
01895                 if (isShort)
01896                 {
01897                     this.ShortOffsets = new ushort[numGlyphs + 1];
01898                     for (int i = 0; i < numGlyphs + 1; i++)
01899                     {
01900                         this.ShortOffsets[i] = sr.ReadUShort();
01901                     }
01902 
01903                     for (int i = 0; i < numGlyphs; i++)
01904                     {
01905                         this.Lengths[i] = 2 * ((uint)this.ShortOffsets[i + 1] -
      (uint)this.ShortOffsets[i]);
01906                     }
01907                 }
01908                 else
01909                 {
01910                     this.IntOffsets = new uint[numGlyphs + 1];
01911                     for (int i = 0; i < numGlyphs + 1; i++)
01912                     {
01913                         this.IntOffsets[i] = sr.ReadUInt();
```

```
01914                     }
01915
01916                     for (int i = 0; i < numGlyphs; i++)
01917                     {
01918                         this.Lengths[i] = this.IntOffsets[i + 1] - this.IntOffsets[i];
01919                     }
01920                 }
01921
01922             }
01923
01924             public byte[] GetBytes()
01925             {
01926                 using (MemoryStream ms = new MemoryStream())
01927                 {
01928
01929                     if (IntOffsets == null)
01930                     {
01931                         for (int i = 0; i < ShortOffsets.Length; i++)
01932                         {
01933                             ms.WriteUShort(ShortOffsets[i]);
01934                         }
01935                     }
01936                     else
01937                     {
01938                         for (int i = 0; i < IntOffsets.Length; i++)
01939                         {
01940                             ms.WriteUInt(IntOffsets[i]);
01941                         }
01942                     }
01943
01944                     return ms.ToArray();
01945                 }
01946             }
01947         }
01948
01949         internal class TrueTypeRawTable :  ITrueTypeTable
01950         {
01951             public byte[] Data { get; }
01952
01953             public TrueTypeRawTable(Stream sr, uint length)
01954             {
01955                 this.Data = new byte[length];
01956                 sr.Read(this.Data, 0, (int)length);
01957             }
01958
01959             public byte[] GetBytes()
01960             {
01961                 return Data;
01962             }
01963         }
01964
01965         internal bool HasCmap4Table()
01966         {
01967             foreach (ICmapTable cmap in ((TrueTypeCmapTable)Tables["cmap"]).ActualCmapTables)
01968             {
01969                 if (cmap is CmapTable4)
01970                 {
01971                     return true;
01972                 }
01973
01974             }
01975             return false;
01976         }
01977
01978 /// <summary>
01979 /// Obtains the font family name from the TrueType file.
01980 /// </summary>
01981 /// <returns>The font family name, if available; <see langword="null"/> otherwise.</returns>
01982         public string GetFontFamilyName()
01983         {
01984             TrueTypeNameTable name = (TrueTypeNameTable)this.Tables["name"];
01985
01986             for (int i = 0; i < name.Count; i++)
01987             {
01988                 if (name.NameRecords[i].NameID == 16)
01989                 {
01990                     return name.Name[i];
01991                 }
01992             }
01993
01994             for (int i = 0; i < name.Count; i++)
01995             {
01996                 if (name.NameRecords[i].NameID == 1)
01997                 {
01998                     return name.Name[i];
01999                 }
02000             }
```

```
02001
02002             return null;
02003         }
02004
02005 /// <summary>
02006 /// Obtains the full font family name from the TrueType file.
02007 /// </summary>
02008 /// <returns>The full font family name, if available; <see langword="null"/> otherwise.</returns>
02009         public string GetFullFontFamilyName()
02010         {
02011             TrueTypeNameTable name = (TrueTypeNameTable)this.Tables["name"];
02012
02013             for (int i = 0; i < name.Count; i++)
02014             {
02015                 if (name.NameRecords[i].NameID == 4)
02016                 {
02017                     return name.Name[i];
02018                 }
02019             }
02020
02021             for (int i = 0; i < name.Count; i++)
02022             {
02023                 if (name.NameRecords[i].NameID == 6)
02024                 {
02025                     return name.Name[i];
02026                 }
02027             }
02028
02029             return null;
02030         }
02031
02032
02033 /// <summary>
02034 /// Obtains the PostScript font name from the TrueType file.
02035 /// </summary>
02036 /// <returns>The PostScript font name, if available; <see langword="null"/> otherwise.</returns>
02037         public string GetFontName()
02038         {
02039             TrueTypeNameTable name = (TrueTypeNameTable)this.Tables["name"];
02040
02041             for (int i = 0; i < name.Count; i++)
02042             {
02043                 if (name.NameRecords[i].NameID == 6)
02044                 {
02045                     return name.Name[i];
02046                 }
02047             }
02048
02049             return null;
02050         }
02051
02052 /// <summary>
02053 /// Returns the index of the first character glyph represented by the font.
02054 /// </summary>
02055 /// <returns>The index of the first character glyph represented by the font.</returns>
02056         public ushort GetFirstCharIndex()
02057         {
02058             TrueTypeOS2Table os2 = (TrueTypeOS2Table)this.Tables["OS/2"];
02059
02060             return os2.FsFirstCharIndex;
02061         }
02062
02063 /// <summary>
02064 /// Returns the index of the last character glyph represented by the font.
02065 /// </summary>
02066 /// <returns>The index of the last character glyph represented by the font.</returns>
02067         public ushort GetLastCharIndex()
02068         {
02069             TrueTypeOS2Table os2 = (TrueTypeOS2Table)this.Tables["OS/2"];
02070
02071             return os2.FsLastCharIndex;
02072         }
02073
02074
02075 /// <summary>
02076 /// Determines whether the typeface is Italic or Oblique or not.
02077 /// </summary>
02078 /// <returns>A <see cref="bool"/> indicating whether the typeface is Italic or Oblique or
      not.</returns>
02079         public bool IsItalic()
02080         {
02081             TrueTypeOS2Table os2 = (TrueTypeOS2Table)this.Tables["OS/2"];
02082
02083             return (os2.FsSelection & 1) == 1;
02084         }
02085
02086 /// <summary>
```

```
02087 /// Determines whether the typeface is Oblique or not.
02088 /// </summary>
02089 /// <returns>A <see cref="bool"/> indicating whether the typeface is Oblique or not.</returns>
02090         public bool IsOblique()
02091         {
02092             TrueTypeOS2Table os2 = (TrueTypeOS2Table)this.Tables["OS/2"];
02093 
02094             return (os2.FsSelection & 512) != 0;
02095         }
02096 
02097 /// <summary>
02098 /// Determines whether the typeface is Bold or not.
02099 /// </summary>
02100 /// <returns>A <see cref="bool"/> indicating whether the typeface is Bold or not</returns>
02101         public bool IsBold()
02102         {
02103             TrueTypeOS2Table os2 = (TrueTypeOS2Table)this.Tables["OS/2"];
02104 
02105             return (os2.FsSelection & 32) != 0;
02106         }
02107 
02108 /// <summary>
02109 /// Determines whether the typeface is fixed-pitch (aka monospaces) or not.
02110 /// </summary>
02111 /// <returns>A <see cref="bool"/> indicating whether the typeface is fixed-pitch (aka monospaces) or
      not.</returns>
02112         public bool IsFixedPitch()
02113         {
02114             TrueTypeOS2Table os2 = (TrueTypeOS2Table)this.Tables["OS/2"];
02115 
02116             return os2.Panose.BProportion == 9;
02117         }
02118 
02119 /// <summary>
02120 /// Determines whether the typeface is serifed or not.
02121 /// </summary>
02122 /// <returns>A <see cref="bool"/> indicating whether the typeface is serifed or not.</returns>
02123         public bool IsSerif()
02124         {
02125             TrueTypeOS2Table os2 = (TrueTypeOS2Table)this.Tables["OS/2"];
02126 
02127             return os2.SFamilyClass == 1 || os2.SFamilyClass == 2 || os2.SFamilyClass == 3 ||
      os2.SFamilyClass == 4 || os2.SFamilyClass == 5 || os2.SFamilyClass == 7;
02128         }
02129 
02130 /// <summary>
02131 /// Determines whether the typeface is a script typeface or not.
02132 /// </summary>
02133 /// <returns>A <see cref="bool"/> indicating whether the typeface is a script typeface or
      not.</returns>
02134         public bool IsScript()
02135         {
02136             TrueTypeOS2Table os2 = (TrueTypeOS2Table)this.Tables["OS/2"];
02137 
02138             return os2.SFamilyClass == 10;
02139         }
02140 
02141 /// <summary>
02142 /// Determines the index of the glyph corresponding to a certain character.
02143 /// </summary>
02144 /// <param name="glyph">The character whose glyph is sought.</param>
02145 /// <returns>The index of the glyph in the TrueType file.</returns>
02146         public int GetGlyphIndex(char glyph)
02147         {
02148             foreach (ICmapTable cmap in ((TrueTypeCmapTable)Tables["cmap"]).ActualCmapTables)
02149             {
02150                 if (cmap is CmapTable4)
02151                 {
02152                     return cmap.GetGlyphIndex(glyph);
02153                 }
02154             }
02155 
02156             foreach (ICmapTable cmap in ((TrueTypeCmapTable)Tables["cmap"]).ActualCmapTables)
02157             {
02158                 if (cmap is CmapTable0)
02159                 {
02160                     return cmap.GetGlyphIndex(glyph);
02161                 }
02162             }
02163 
02164             return -1;
02165         }
02166 
02167         internal int GetGlyphWidth(int glyphIndex)
02168         {
02169             if (((TrueTypeHmtxTable)this.Tables["hmtx"]).HMetrics.Length > glyphIndex)
02170             {
```

```
02171                 return ((TrueTypeHmtxTable)this.Tables["hmtx"]).HMetrics[glyphIndex].AdvanceWidth;
02172             }
02173             else
02174             {
02175                 return ((TrueTypeHmtxTable)this.Tables["hmtx"]).HMetrics.Last().AdvanceWidth;
02176             }
02177         }
02178 
02179         internal LongHorFixed GetGlyphMetrics(int glyphIndex)
02180         {
02181             if (((TrueTypeHmtxTable)this.Tables["hmtx"]).HMetrics.Length > glyphIndex)
02182             {
02183                 return ((TrueTypeHmtxTable)this.Tables["hmtx"]).HMetrics[glyphIndex];
02184             }
02185             else
02186             {
02187                 return new
      LongHorFixed(((TrueTypeHmtxTable)this.Tables["hmtx"]).HMetrics.Last().AdvanceWidth,
      ((TrueTypeHmtxTable)this.Tables["hmtx"]).LeftSideBearing[glyphIndex -
      ((TrueTypeHmtxTable)this.Tables["hmtx"]).HMetrics.Length]);
02188             }
02189         }
02190 
02191 /// <summary>
02192 /// Get the path that describes the shape of a glyph.
02193 /// </summary>
02194 /// <param name="glyphIndex">The index of the glyph whose path is sought.</param>
02195 /// <param name="size">The font size to be used for the font coordinates.</param>
02196 /// <returns>An array of contours, each of which is itself an array of TrueType points.</returns>
02197         public TrueTypePoint[][] GetGlyphPath(int glyphIndex, double size)
02198         {
02199             return ((TrueTypeGlyfTable)this.Tables["glyf"]).Glyphs[glyphIndex].GetGlyphPath(size,
      ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm, ((TrueTypeGlyfTable)this.Tables["glyf"]).Glyphs);
02200         }
02201 
02202 /// <summary>
02203 /// Get the path that describes the shape of a glyph.
02204 /// </summary>
02205 /// <param name="glyph">The glyph whose path is sought.</param>
02206 /// <param name="size">The font size to be used for the font coordinates.</param>
02207 /// <returns>An array of contours, each of which is itself an array of TrueType points.</returns>
02208         public TrueTypePoint[][] GetGlyphPath(char glyph, double size)
02209         {
02210             return
      ((TrueTypeGlyfTable)this.Tables["glyf"]).Glyphs[GetGlyphIndex(glyph)].GetGlyphPath(size,
      ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm, ((TrueTypeGlyfTable)this.Tables["glyf"]).Glyphs);
02211         }
02212 
02213 /// <summary>
02214 /// Computes the advance width of a glyph, in thousandths of em unit.
02215 /// </summary>
02216 /// <param name="glyph">The glyph whose advance width is to be computed.</param>
02217 /// <returns>The advance width of the glyph in thousandths of em unit.</returns>
02218         public double Get1000EmGlyphWidth(char glyph)
02219         {
02220             int num = (int)glyph;
02221             if (num < 256)
02222             {
02223                 return GlyphWidthsCache[num];
02224             }
02225             else
02226             {
02227                 return Get1000EmGlyphWidth(GetGlyphIndex(glyph));
02228             }
02229         }
02230 
02231 /// <summary>
02232 /// Computes the advance width of a glyph, in thousandths of em unit.
02233 /// </summary>
02234 /// <param name="glyphIndex">The index of the glyph whose advance width is to be computed.</param>
02235 /// <returns>The advance width of the glyph in thousandths of em unit.</returns>
02236         public double Get1000EmGlyphWidth(int glyphIndex)
02237         {
02238             int w = GetGlyphWidth(glyphIndex);
02239 
02240             return w * 1000 / ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm;
02241         }
02242 
02243 /// <summary>
02244 /// Computes the font's Win ascent, in thousandths of em unit.
02245 /// </summary>
02246 /// <returns>The font's Win ascent in thousandths of em unit.</returns>
02247         public double Get1000EmWinAscent()
02248         {
02249             TrueTypeOS2Table os2 = ((TrueTypeOS2Table)this.Tables["OS/2"]);
02250 
02251             bool useTypoMetrics = (os2.FsSelection & 128) != 0;
```

```
02252
02253             if (!useTypoMetrics)
02254             {
02255                 return ((TrueTypeOS2Table)this.Tables["OS/2"]).UsWinAscent * 1000.0 /
      ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm;
02256             }
02257             else
02258             {
02259                 return ((TrueTypeOS2Table)this.Tables["OS/2"]).STypoAscender * 1000.0 /
      ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm;
02260             }
02261         }
02262
02263 /// <summary>
02264 /// Computes the font ascent, in thousandths of em unit.
02265 /// </summary>
02266 /// <returns>The font ascent in thousandths of em unit.</returns>
02267         public double Get1000EmAscent()
02268         {
02269             return ((TrueTypeHHeaTable)this.Tables["hhea"]).Ascent * 1000.0 /
      ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm;
02270         }
02271
02272
02273 /// <summary>
02274 /// Computes the font descent, in thousandths of em unit.
02275 /// </summary>
02276 /// <returns>The font descent in thousandths of em unit.</returns>
02277         public double Get1000EmDescent()
02278         {
02279             return ((TrueTypeHHeaTable)this.Tables["hhea"]).Descent * 1000.0 /
      ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm;
02280         }
02281
02282 /// <summary>
02283 /// Computes the maximum height over the baseline of the font, in thousandths of em unit.
02284 /// </summary>
02285 /// <returns>The maximum height over the baseline of the font in thousandths of em unit.</returns>
02286         public double Get1000EmYMax()
02287         {
02288             return ((TrueTypeHeadTable)this.Tables["head"]).YMax * 1000.0 /
      ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm;
02289         }
02290
02291 /// <summary>
02292 /// Computes the maximum depth below the baseline of the font, in thousandths of em unit.
02293 /// </summary>
02294 /// <returns>The maximum depth below the baseline of the font in thousandths of em unit.</returns>
02295         public double Get1000EmYMin()
02296         {
02297             return ((TrueTypeHeadTable)this.Tables["head"]).YMin * 1000.0 /
      ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm;
02298         }
02299
02300 /// <summary>
02301 /// Computes the maximum distance to the right of the glyph origin of the font, in thousandths of em
      unit.
02302 /// </summary>
02303 /// <returns>The maximum distance to the right of the glyph origin of the font in thousandths of em
      unit.</returns>
02304         public double Get1000EmXMax()
02305         {
02306             return ((TrueTypeHeadTable)this.Tables["head"]).XMax * 1000.0 /
      ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm;
02307         }
02308
02309 /// <summary>
02310 /// Computes the maximum distance to the left of the glyph origin of the font, in thousandths of em
      unit.
02311 /// </summary>
02312 /// <returns>The maximum distance to the left of the glyph origin of the font in thousandths of em
      unit.</returns>
02313         public double Get1000EmXMin()
02314         {
02315             return ((TrueTypeHeadTable)this.Tables["head"]).XMin * 1000.0 /
      ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm;
02316         }
02317
02318 /// <summary>
02319 /// Represents the left- and right-side bearings of a glyph.
02320 /// </summary>
02321         public struct Bearings
02322         {
02323 /// <summary>
02324 /// The left-side bearing of the glyph.
02325 /// </summary>
02326             public int LeftSideBearing;
```

```
02327
02328 /// <summary>
02329 /// The right-side bearing of the glyph.
02330 /// </summary>
02331             public int RightSideBearing;
02332
02333             internal Bearings(int lsb, int rsb)
02334             {
02335                 LeftSideBearing = lsb;
02336                 RightSideBearing = rsb;
02337             }
02338         }
02339
02340         internal Bearings Get1000EmGlyphBearings(int glyphIndex)
02341         {
02342             LongHorFixed metrics = GetGlyphMetrics(glyphIndex);
02343
02344             int lsb = metrics.LeftSideBearing;
02345
02346             Glyph glyph = ((TrueTypeGlyfTable)this.Tables["glyf"]).Glyphs[glyphIndex];
02347
02348             int rsb = metrics.AdvanceWidth - (lsb + glyph.XMax - glyph.XMin);
02349
02350             return new Bearings(lsb * 1000 / ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm, rsb
      * 1000 / ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm);
02351         }
02352
02353
02354 /// <summary>
02355 /// Computes the left- and right- side bearings of a glyph, in thousandths of em unit.
02356 /// </summary>
02357 /// <param name="glyph">The glyph whose bearings are to be computed.</param>
02358 /// <returns>The left- and right- side bearings of the glyph in thousandths of em unit</returns>
02359         public Bearings Get1000EmGlyphBearings(char glyph)
02360         {
02361             int num = (int)glyph;
02362
02363             if (num < 256)
02364             {
02365                 return BearingsCache[num];
02366             }
02367             else
02368             {
02369                 return Get1000EmGlyphBearings(GetGlyphIndex(glyph));
02370             }
02371         }
02372
02373 /// <summary>
02374 /// Represents the maximum heigth above and depth below the baseline of a glyph.
02375 /// </summary>
02376         public struct VerticalMetrics
02377         {
02378 /// <summary>
02379 /// The maximum depth below the baseline of the glyph.
02380 /// </summary>
02381             public int YMin;
02382
02383 /// <summary>
02384 /// The maximum height above the baseline of the glyph.
02385 /// </summary>
02386             public int YMax;
02387
02388             internal VerticalMetrics(int yMin, int yMax)
02389             {
02390                 this.YMin = yMin;
02391                 this.YMax = yMax;
02392             }
02393         }
02394
02395         internal VerticalMetrics Get1000EmGlyphVerticalMetrics(int glyphIndex)
02396         {
02397             Glyph glyph = ((TrueTypeGlyfTable)this.Tables["glyf"]).Glyphs[glyphIndex];
02398
02399             return new VerticalMetrics(glyph.YMin * 1000 /
      ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm, glyph.YMax * 1000 /
      ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm);
02400         }
02401
02402 /// <summary>
02403 /// Computes the vertical metrics of a glyph, in thousandths of em unit.
02404 /// </summary>
02405 /// <param name="glyph">The glyph whose vertical metrics are to be computed.</param>
02406 /// <returns>The vertical metrics of a glyph, in thousandths of em unit.</returns>
02407         public VerticalMetrics Get1000EmGlyphVerticalMetrics(char glyph)
02408         {
02409             int num = (int)glyph;
02410             if (num < 256)
```

```
02411             {
02412                 return VerticalMetricsCache[num];
02413             }
02414             else
02415             {
02416                 return Get1000EmGlyphVerticalMetrics(GetGlyphIndex(glyph));
02417             }
02418         }
02419 
02420 /// <summary>
02421 /// Computes the distance of the top of the underline from the baseline, in thousandths of em unit.
02422 /// </summary>
02423 /// <returns>The distance of the top of the underline from the baseline, in thousandths of em
      unit.</returns>
02424         public double Get1000EmUnderlinePosition()
02425         {
02426             if (this.Tables.TryGetValue("post", out ITrueTypeTable table) && table is
      TrueTypePostTable post)
02427             {
02428                 return post.UnderlinePosition * 1000.0 /
      ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm;
02429             }
02430             else
02431             {
02432                 return double.NaN;
02433             }
02434         }
02435 
02436 /// <summary>
02437 /// Computes the thickness of the underline, in thousandths of em unit.
02438 /// </summary>
02439 /// <returns>The thickness of the underline, in thousandths of em unit.</returns>
02440         public double Get1000EmUnderlineThickness()
02441         {
02442             if (this.Tables.TryGetValue("post", out ITrueTypeTable table) && table is
      TrueTypePostTable post)
02443             {
02444                 return post.UnderlineThickness * 1000.0 /
      ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm;
02445             }
02446             else
02447             {
02448                 return double.NaN;
02449             }
02450         }
02451 
02452 /// <summary>
02453 /// Returns the number of units for each em in the font file.  Multiplying any glyph measurement by
      this
02454 /// size should generally lead to an integer value.
02455 /// </summary>
02456 /// <returns>The number of units for each em in the font file.</returns>
02457         public int GetUnitsPerEm()
02458         {
02459             return ((TrueTypeHeadTable)Tables["head"]).UnitsPerEm;
02460         }
02461 
02462 /// <summary>
02463 /// Computes the italic angle for the current font, in counter-clockwise degrees from the vertical.
      This is negative for text that leans forwards.
02464 /// </summary>
02465 /// <returns></returns>
02466         public double GetItalicAngle()
02467         {
02468             if (this.Tables.TryGetValue("post", out ITrueTypeTable table) && table is
      TrueTypePostTable post)
02469             {
02470                 return post.ItalicAngle.Bits / Math.Pow(2, post.ItalicAngle.BitShifts);
02471             }
02472             else
02473             {
02474                 return double.NaN;
02475             }
02476         }
02477 
02478 /// <summary>
02479 /// Computes the intersections between an underline at the specified position and thickness and a
      glyph, in thousandths of em units.
02480 /// </summary>
02481 /// <param name="glyph">The glyph whose intersections with the underline will be computed.</param>
02482 /// <param name="position">The distance of the top of the underline from the baseline, in thousandths
      of em unit.</param>
02483 /// <param name="thickness">The thickness of the underline, in thousandths of em unit.</param>
02484 /// <returns>If the underline does not intersect the glyph, this method returns <see langword="null"
      />.  Otherwise, it returns an array
02485 /// containing two elements, representing the horizontal coordinates of the leftmost and rightmost
      intersection points.</returns>
```

```
02486         public double[] Get1000EmUnderlineIntersections(char glyph, double position, double thickness)
02487         {
02488             List<double> intersections = new List<double>();
02489 
02490             TrueTypePoint[][] glyphPaths = GetGlyphPath(glyph, 1000);
02491 
02492             for (int j = 0; j < glyphPaths.Length; j++)
02493             {
02494                 double[] currPoint = new double[] { glyphPaths[j][0].X, -glyphPaths[j][0].Y };
02495 
02496                 for (int k = 1; k < glyphPaths[j].Length; k++)
02497                 {
02498                     if (glyphPaths[j][k].IsOnCurve)
02499                     {
02500                         if (-glyphPaths[j][k].Y - currPoint[1] != 0)
02501                         {
02502                             double t = (position - currPoint[1]) / (-glyphPaths[j][k].Y -
      currPoint[1]);
02503                             if (t >= 0 && t <= 1)
02504                             {
02505                                 intersections.Add(currPoint[0] + t * (glyphPaths[j][k].X -
      currPoint[0]));
02506                             }
02507 
02508                             t = (position + thickness - currPoint[1]) / (-glyphPaths[j][k].Y -
      currPoint[1]);
02509                             if (t >= 0 && t <= 1)
02510                             {
02511                                 intersections.Add(currPoint[0] + t * (glyphPaths[j][k].X -
      currPoint[0]));
02512                             }
02513                         }
02514                         else
02515                         {
02516                             if (position <= currPoint[1] && position + thickness >= currPoint[1])
02517                             {
02518                                 intersections.Add(currPoint[0]);
02519                                 intersections.Add(glyphPaths[j][k].X);
02520                             }
02521                         }
02522 
02523                         currPoint = new double[] { glyphPaths[j][k].X, -glyphPaths[j][k].Y };
02524                     }
02525                     else
02526                     {
02527                         double[] ctrlPoint = new double[] { glyphPaths[j][k].X, -glyphPaths[j][k].Y };
02528                         double[] endPoint = new double[] { glyphPaths[j][k + 1].X, -glyphPaths[j][k +
      1].Y };
02529 
02530                         double a = currPoint[1] - 2 * ctrlPoint[1] + endPoint[1];
02531 
02532                         if (Math.Abs(a) < 1e-7)
02533                         {
02534                             k++;
02535 
02536                             if (-glyphPaths[j][k].Y - currPoint[1] != 0)
02537                             {
02538                                 double t = (position - currPoint[1]) / (-glyphPaths[j][k].Y -
      currPoint[1]);
02539                                 if (t >= 0 && t <= 1)
02540                                 {
02541                                     intersections.Add(currPoint[0] + t * (glyphPaths[j][k].X -
      currPoint[0]));
02542                                 }
02543 
02544                                 t = (position + thickness - currPoint[1]) / (-glyphPaths[j][k].Y -
      currPoint[1]);
02545                                 if (t >= 0 && t <= 1)
02546                                 {
02547                                     intersections.Add(currPoint[0] + t * (glyphPaths[j][k].X -
      currPoint[0]));
02548                                 }
02549                             }
02550                             else
02551                             {
02552                                 if (position <= currPoint[1] && position + thickness >= currPoint[1])
02553                                 {
02554                                     intersections.Add(currPoint[0]);
02555                                     intersections.Add(glyphPaths[j][k].X);
02556                                 }
02557                             }
02558                         }
02559                         else
02560                         {
02561                             double[] ts = new double[4];
02562 
02563                             ts[0] = (currPoint[1] - ctrlPoint[1] - Math.Sqrt(position * a +
```

```
ctrlPoint[1] * ctrlPoint[1] - currPoint[1] * endPoint[1])) / a;
02564                           ts[1] = (currPoint[1] - ctrlPoint[1] + Math.Sqrt(position * a +
ctrlPoint[1] * ctrlPoint[1] - currPoint[1] * endPoint[1])) / a;
02565
02566                           ts[2] = (currPoint[1] - ctrlPoint[1] - Math.Sqrt((position + thickness) *
a + ctrlPoint[1] * ctrlPoint[1] - currPoint[1] * endPoint[1])) / a;
02567                           ts[3] = (currPoint[1] - ctrlPoint[1] + Math.Sqrt((position + thickness) *
a + ctrlPoint[1] * ctrlPoint[1] - currPoint[1] * endPoint[1])) / a;
02568
02569                           double minT = double.MaxValue;
02570                           double maxT = double.MinValue;
02571                           bool found = false;
02572
02573                           for (int i = 0; i < 4; i++)
02574                           {
02575                               if (!double.IsNaN(ts[i]) && ts[i] >= 0 && ts[i] <= 1)
02576                               {
02577                                   minT = Math.Min(minT, ts[i]);
02578                                   maxT = Math.Max(maxT, ts[i]);
02579
02580                                   found = true;
02581                               }
02582                           }
02583
02584                           if (found)
02585                           {
02586                               double critT = (currPoint[0] - ctrlPoint[0]) / (currPoint[0] - 2 *
ctrlPoint[0] + endPoint[0]);
02587
02588                               if (critT >= minT && critT <= maxT)
02589                               {
02590                                   intersections.Add((1 - critT) * (1 - critT) * currPoint[0] + 2 *
critT * (1 - critT) * ctrlPoint[0] + critT * critT * endPoint[0]);
02591                               }
02592
02593                               intersections.Add((1 - minT) * (1 - minT) * currPoint[0] + 2 * minT *
(1 - minT) * ctrlPoint[0] + minT * minT * endPoint[0]);
02594                               intersections.Add((1 - maxT) * (1 - maxT) * currPoint[0] + 2 * maxT *
(1 - maxT) * ctrlPoint[0] + maxT * maxT * endPoint[0]);
02595                           }
02596
02597                           k++;
02598                       }
02599
02600                       currPoint = endPoint;
02601                   }
02602               }
02603           }
02604
02605           if (intersections.Count > 1)
02606           {
02607               double[] tbr = new double[2] { double.MaxValue, double.MinValue };
02608
02609               for (int i = 0; i < intersections.Count; i++)
02610               {
02611                   if (intersections[i] <= tbr[0])
02612                   {
02613                       tbr[0] = intersections[i];
02614                   }
02615
02616                   if (intersections[i] >= tbr[1])
02617                   {
02618                       tbr[1] = intersections[i];
02619                   }
02620               }
02621
02622               return tbr;
02623           }
02624           else
02625           {
02626               return null;
02627           }
02628       }
02629
02630 /// <summary>
02631 /// Gets the kerning between two glyphs.
02632 /// </summary>
02633 /// <param name="glyph1">The first glyph of the kerning pair.</param>
02634 /// <param name="glyph2">The second glyph of the kerning pair.</param>
02635 /// <returns>A <see cref="PairKerning"/> object containing information about how the position of each
glyphs should be altered.</returns>
02636       public PairKerning Get1000EmKerning(char glyph1, char glyph2)
02637       {
02638           int glyph1Index = this.GetGlyphIndex(glyph1);
02639           int glyph2Index = this.GetGlyphIndex(glyph2);
02640
02641           return Get1000EmKerning(glyph1Index, glyph2Index);
```

```
02642         }
02643
02644 /// <summary>
02645 /// Gets the kerning between two glyphs.
02646 /// </summary>
02647 /// <param name="glyph1Index">The index of the first glyph of the kerning pair.</param>
02648 /// <param name="glyph2Index">The index of the second glyph of the kerning pair.</param>
02649 /// <returns>A <see cref="PairKerning"/> object containing information about how the position of each
      glyphs should be altered.</returns>
02650         public PairKerning Get1000EmKerning(int glyph1Index, int glyph2Index)
02651         {
02652             if (this.Tables.TryGetValue("GPOS", out ITrueTypeTable table) && table is
      TrueTypeGPOSTable gpos)
02653             {
02654                 PairKerning kerning = gpos.GetKerning(glyph1Index, glyph2Index);
02655
02656                 if (kerning != null)
02657                 {
02658                     double units = ((TrueTypeHeadTable)this.Tables["head"]).UnitsPerEm;
02659
02660                     return new PairKerning(new Point(kerning.Glyph1Placement.X * 1000 / units,
      kerning.Glyph1Placement.Y * 1000 / units),
02661                         new Point(kerning.Glyph1Advance.X * 1000 / units, kerning.Glyph1Advance.Y *
      1000 / units),
02662                         new Point(kerning.Glyph2Placement.X * 1000 / units, kerning.Glyph2Placement.Y
      * 1000 / units),
02663                         new Point(kerning.Glyph2Advance.X * 1000 / units, kerning.Glyph2Advance.Y *
      1000 / units));
02664                 }
02665                 else
02666                 {
02667                     return null;
02668                 }
02669             }
02670             else
02671             {
02672                 return null;
02673             }
02674         }
02675
02676         internal interface ITrueTypeTable
02677         {
02678             byte[] GetBytes();
02679         }
02680
02681         internal interface ICmapTable
02682         {
02683             ushort Format { get; }
02684             ushort Length { get; }
02685             ushort Language { get; }
02686
02687             int GetGlyphIndex(char glyph);
02688
02689             byte[] GetBytes();
02690         }
02691
02692         internal class CmapTable4 :  ICmapTable
02693         {
02694             public ushort Format { get; set; }
02695             public ushort Length { get; set; }
02696             public ushort Language { get; set; }
02697             public ushort SegCountX2 { get; set; }
02698             public ushort SearchRange { get; set; }
02699             public ushort EntrySelector { get; set; }
02700             public ushort RangeShift { get; set; }
02701             public ushort[] EndCode { get; set; }
02702             public ushort ReservedPad { get; set; }
02703             public ushort[] StartCode { get; set; }
02704             public ushort[] IdDelta { get; set; }
02705             public ushort[] IdRangeOffset { get; set; }
02706             public ushort[] GlyphIndexArray { get; set; }
02707
02708             public CmapTable4() { }
02709
02710             public CmapTable4(ushort format, ushort length, ushort language, Stream sr)
02711             {
02712                 this.Format = format;
02713                 this.Length = length;
02714                 this.Language = language;
02715                 this.SegCountX2 = sr.ReadUShort();
02716
02717                 int segCount = this.SegCountX2 / 2;
02718
02719                 this.SearchRange = sr.ReadUShort();
02720                 this.EntrySelector = sr.ReadUShort();
02721                 this.RangeShift = sr.ReadUShort();
02722
```

```
02723                 this.EndCode = new ushort[segCount];
02724                 for (int i = 0; i < segCount; i++)
02725                 {
02726                     this.EndCode[i] = sr.ReadUShort();
02727                 }
02728 
02729                 this.ReservedPad = sr.ReadUShort();
02730 
02731                 this.StartCode = new ushort[segCount];
02732                 for (int i = 0; i < segCount; i++)
02733                 {
02734                     this.StartCode[i] = sr.ReadUShort();
02735                 }
02736 
02737                 this.IdDelta = new ushort[segCount];
02738                 for (int i = 0; i < segCount; i++)
02739                 {
02740                     this.IdDelta[i] = sr.ReadUShort();
02741                 }
02742 
02743                 this.IdRangeOffset = new ushort[segCount];
02744                 for (int i = 0; i < segCount; i++)
02745                 {
02746                     this.IdRangeOffset[i] = sr.ReadUShort();
02747                 }
02748 
02749                 int numGlyphIndices = (this.Length - 16 + 8 * segCount) / 2;
02750 
02751                 this.GlyphIndexArray = new ushort[numGlyphIndices];
02752 
02753                 for (int i = 0; i < numGlyphIndices; i++)
02754                 {
02755                     this.GlyphIndexArray[i] = sr.ReadUShort();
02756                 }
02757             }
02758 
02759             public int GetGlyphIndex(char glyph)
02760             {
02761                 int code = (int)glyph;
02762 
02763                 int endCodeInd = -1;
02764 
02765                 for (int i = 0; i < EndCode.Length; i++)
02766                 {
02767                     if (EndCode[i] >= code)
02768                     {
02769                         endCodeInd = i;
02770                         break;
02771                     }
02772                 }
02773 
02774                 if (StartCode[endCodeInd] <= code)
02775                 {
02776                     if (IdRangeOffset[endCodeInd] != 0)
02777                     {
02778                         int glyphIndexIndex = IdRangeOffset[endCodeInd] / 2 + (code -
    StartCode[endCodeInd]) - (IdRangeOffset.Length - endCodeInd);
02779 
02780                         if (GlyphIndexArray[glyphIndexIndex] != 0)
02781                         {
02782                             return (IdDelta[endCodeInd] + GlyphIndexArray[glyphIndexIndex]) % 65536;
02783                         }
02784                         else
02785                         {
02786                             return 0;
02787                         }
02788                     }
02789                     else
02790                     {
02791                         return (code + IdDelta[endCodeInd]) % 65536;
02792                     }
02793                 }
02794                 else
02795                 {
02796                     return 0;
02797                 }
02798             }
02799 
02800             public byte[] GetBytes()
02801             {
02802                 using (MemoryStream ms = new MemoryStream())
02803                 {
02804                     ms.WriteUShort(this.Format);
02805                     ms.WriteUShort(this.Length);
02806                     ms.WriteUShort(this.Language);
02807                     ms.WriteUShort(this.SegCountX2);
02808                     ms.WriteUShort(this.SearchRange);
```

```
02809                   ms.WriteUShort(this.EntrySelector);
02810                   ms.WriteUShort(this.RangeShift);
02811                   for (int i = 0; i < this.EndCode.Length; i++)
02812                   {
02813                       ms.WriteUShort(this.EndCode[i]);
02814                   }
02815                   ms.WriteUShort(this.ReservedPad);
02816
02817                   for (int i = 0; i < this.StartCode.Length; i++)
02818                   {
02819                       ms.WriteUShort(this.StartCode[i]);
02820                   }
02821                   for (int i = 0; i < this.IdDelta.Length; i++)
02822                   {
02823                       ms.WriteUShort(this.IdDelta[i]);
02824                   }
02825                   for (int i = 0; i < this.IdRangeOffset.Length; i++)
02826                   {
02827                       ms.WriteUShort(this.IdRangeOffset[i]);
02828                   }
02829                   for (int i = 0; i < this.GlyphIndexArray.Length; i++)
02830                   {
02831                       ms.WriteUShort(this.GlyphIndexArray[i]);
02832                   }
02833
02834                   return ms.ToArray();
02835               }
02836           }
02837       }
02838
02839       internal class CmapTable0 :  ICmapTable
02840       {
02841           public ushort Format { get; }
02842           public ushort Length { get; }
02843           public ushort Language { get; }
02844
02845           public byte[] GlyphIndexArray { get; }
02846
02847           public CmapTable0(ushort format, ushort length, ushort language, byte[] glyphIndexArray)
02848           {
02849               this.Format = format;
02850               this.Length = length;
02851               this.Language = language;
02852               this.GlyphIndexArray = glyphIndexArray;
02853           }
02854
02855           public int GetGlyphIndex(char glyph)
02856           {
02857               return GlyphIndexArray[(byte)glyph];
02858           }
02859
02860           public byte[] GetBytes()
02861           {
02862               using (MemoryStream ms = new MemoryStream())
02863               {
02864                   ms.WriteUShort(this.Format);
02865                   ms.WriteUShort(this.Length);
02866                   ms.WriteUShort(this.Language);
02867                   ms.Write(this.GlyphIndexArray, 0, this.GlyphIndexArray.Length);
02868                   return ms.ToArray();
02869               }
02870           }
02871       }
02872
02873       internal struct CmapSubTable
02874       {
02875           public ushort PlatformID;
02876           public ushort PlatformSpecificID;
02877           public uint Offset;
02878
02879           public CmapSubTable(ushort platformID, ushort platformSpecificID, uint offset)
02880           {
02881               this.PlatformID = platformID;
02882               this.PlatformSpecificID = platformSpecificID;
02883               this.Offset = offset;
02884           }
02885       }
02886
02887       internal class TrueTypeCmapTable :  ITrueTypeTable
02888       {
02889           public ushort Version;
02890           public ushort NumberSubTables;
02891           public CmapSubTable[] SubTables;
02892           public ICmapTable[] ActualCmapTables;
02893
02894           public byte[] GetBytes()
02895           {
```

```
02896                 using (MemoryStream ms = new MemoryStream())
02897                 {
02898                     ms.WriteUShort(this.Version);
02899                     ms.WriteUShort(this.NumberSubTables);
02900 
02901                     for (int i = 0; i < this.SubTables.Length; i++)
02902                     {
02903                         ms.WriteUShort(this.SubTables[i].PlatformID);
02904                         ms.WriteUShort(this.SubTables[i].PlatformSpecificID);
02905                         ms.WriteUInt(this.SubTables[i].Offset);
02906                     }
02907 
02908                     for (int i = 0; i < this.ActualCmapTables.Length; i++)
02909                     {
02910                         byte[] bytes = this.ActualCmapTables[i].GetBytes();
02911                         ms.Write(bytes, 0, bytes.Length);
02912                     }
02913 
02914                     return ms.ToArray();
02915                 }
02916             }
02917         }
02918 
02919         internal class TrueTypeNameTable :  ITrueTypeTable
02920         {
02921             public byte[] GetBytes()
02922             {
02923                 using (MemoryStream ms = new MemoryStream())
02924                 {
02925                     ms.WriteUShort(this.Format);
02926                     ms.WriteUShort(this.Count);
02927                     ms.WriteUShort(this.StringOffset);
02928 
02929                     for (int i = 0; i < this.NameRecords.Length; i++)
02930                     {
02931                         byte[] bytes = this.NameRecords[i].GetBytes();
02932                         ms.Write(bytes, 0, bytes.Length);
02933                     }
02934 
02935                     ms.Write(this.RawBytes, 0, this.RawBytes.Length);
02936 
02937                     return ms.ToArray();
02938                 }
02939             }
02940 
02941             public ushort Format;
02942             public ushort Count;
02943             public ushort StringOffset;
02944             public NameRecord[] NameRecords;
02945             public string[] Name;
02946             private readonly byte[] RawBytes;
02947 
02948             internal struct NameRecord
02949             {
02950                 public ushort PlatformID;
02951                 public ushort PlatformSpecificID;
02952                 public ushort LanguageID;
02953                 public ushort NameID;
02954                 public ushort Length;
02955                 public ushort Offset;
02956 
02957                 public NameRecord(ushort platformID, ushort platformSpecificID, ushort languageID,
      ushort nameID, ushort length, ushort offset)
02958                 {
02959                     this.PlatformID = platformID;
02960                     this.PlatformSpecificID = platformSpecificID;
02961                     this.LanguageID = languageID;
02962                     this.NameID = nameID;
02963                     this.Length = length;
02964                     this.Offset = offset;
02965                 }
02966 
02967                 public byte[] GetBytes()
02968                 {
02969                     using (MemoryStream ms = new MemoryStream())
02970                     {
02971                         ms.WriteUShort(this.PlatformID);
02972                         ms.WriteUShort(this.PlatformSpecificID);
02973                         ms.WriteUShort(this.LanguageID);
02974                         ms.WriteUShort(this.NameID);
02975                         ms.WriteUShort(this.Length);
02976                         ms.WriteUShort(this.Offset);
02977                         return ms.ToArray();
02978                     }
02979                 }
02980             }
02981 
```

```
02982             internal TrueTypeNameTable(uint tableOffset, Stream sr)
02983             {
02984                 this.Format = sr.ReadUShort();
02985                 this.Count = sr.ReadUShort();
02986                 this.StringOffset = sr.ReadUShort();
02987                 this.NameRecords = new NameRecord[this.Count];
02988                 this.Name = new string[this.Count];
02989 
02990                 int maxOffsetItem = -1;
02991 
02992                 for (int i = 0; i < this.Count; i++)
02993                 {
02994                     this.NameRecords[i] = new NameRecord(sr.ReadUShort(), sr.ReadUShort(),
      sr.ReadUShort(), sr.ReadUShort(), sr.ReadUShort(), sr.ReadUShort());
02995                     if (maxOffsetItem < 0 || this.NameRecords[i].Offset >
      this.NameRecords[maxOffsetItem].Offset || (this.NameRecords[i].Offset ==
      this.NameRecords[maxOffsetItem].Offset && this.NameRecords[i].Length >
      this.NameRecords[maxOffsetItem].Length))
02996                     {
02997                         maxOffsetItem = i;
02998                     }
02999                 }
03000 
03001                 this.RawBytes = new byte[this.NameRecords[maxOffsetItem].Offset +
      this.NameRecords[maxOffsetItem].Length];
03002                 sr.Seek(tableOffset + this.StringOffset, SeekOrigin.Begin);
03003                 sr.Read(this.RawBytes, 0, this.RawBytes.Length);
03004 
03005                 for (int i = 0; i < this.Count; i++)
03006                 {
03007                     sr.Seek(tableOffset + this.NameRecords[i].Offset + this.StringOffset,
      SeekOrigin.Begin);
03008                     byte[] stringBytes = new byte[this.NameRecords[i].Length];
03009                     sr.Read(stringBytes, 0, this.NameRecords[i].Length);
03010 
03011                     if (this.NameRecords[i].PlatformID == 0)
03012                     {
03013                         this.Name[i] = Encoding.BigEndianUnicode.GetString(stringBytes);
03014                     }
03015                     else if (this.NameRecords[i].PlatformID == 1 &&
      this.NameRecords[i].PlatformSpecificID == 0)
03016                     {
03017                         this.Name[i] = GetMacRomanString(stringBytes);
03018                     }
03019                     else if (this.NameRecords[i].PlatformID == 3 &&
      (this.NameRecords[i].PlatformSpecificID == 1 || this.NameRecords[i].PlatformSpecificID == 0))
03020                     {
03021                         this.Name[i] = Encoding.BigEndianUnicode.GetString(stringBytes);
03022                     }
03023                     else
03024                     {
03025                         this.Name[i] = "Unsupported encoding:  " +
      this.NameRecords[i].PlatformID.ToString() + "/" + this.NameRecords[i].PlatformSpecificID.ToString();
03026                     }
03027 
03028                 }
03029             }
03030 
03031             private static readonly char[] MacRomanChars = new char[] { '\u0020', '\u0021', '\u0022',
      '\u0023', '\u0024', '\u0025', '\u0026', '\u0027', '\u0028', '\u0029', '\u002a', '\u002b', '\u002c',
      '\u002d', '\u002e', '\u002f', '\u0030', '\u0031', '\u0032', '\u0033', '\u0034', '\u0035', '\u0036',
      '\u0037', '\u0038', '\u0039', '\u003a', '\u003b', '\u003c', '\u003d', '\u003e', '\u003f', '\u0040',
      '\u0041', '\u0042', '\u0043', '\u0044', '\u0045', '\u0046', '\u0047', '\u0048', '\u0049', '\u004a',
      '\u004b', '\u004c', '\u004d', '\u004e', '\u004f', '\u0050', '\u0051', '\u0052', '\u0053', '\u0054',
      '\u0055', '\u0056', '\u0057', '\u0058', '\u0059', '\u005a', '\u005b', '\u005c', '\u005d', '\u005e',
      '\u005f', '\u0060', '\u0061', '\u0062', '\u0063', '\u0064', '\u0065', '\u0066', '\u0067', '\u0068',
      '\u0069', '\u006a', '\u006b', '\u006c', '\u006d', '\u006e', '\u006f', '\u0070', '\u0071', '\u0072',
      '\u0073', '\u0074', '\u0075', '\u0076', '\u0077', '\u0078', '\u0079', '\u007a', '\u007b', '\u007c',
      '\u007d', '\u007e', '\u007f', '\u00c4', '\u00c5', '\u00c7', '\u00c9', '\u00d1', '\u00d6', '\u00dc',
      '\u00e1', '\u00e0', '\u00e2', '\u00e4', '\u00e3', '\u00e5', '\u00e7', '\u00e9', '\u00e8', '\u00ea',
      '\u00eb', '\u00ed', '\u00ec', '\u00ee', '\u00ef', '\u00f1', '\u00f3', '\u00f2', '\u00f4', '\u00f6',
      '\u00f5', '\u00fa', '\u00f9', '\u00fb', '\u00fc', '\u2020', '\u00b0', '\u00a2', '\u00a3', '\u00a7',
      '\u2022', '\u00b6', '\u00df', '\u00ae', '\u00a9', '\u2122', '\u00b4', '\u00a8', '\u2260', '\u00c6',
      '\u00d8', '\u221e', '\u00b1', '\u2264', '\u2265', '\u00a5', '\u00b5', '\u2202', '\u2211', '\u220f',
      '\u03c0', '\u222b', '\u00aa', '\u00ba', '\u03a9', '\u00e6', '\u00f8', '\u00bf', '\u00a1', '\u00ac',
      '\u221a', '\u0192', '\u2248', '\u2206', '\u00ab', '\u00bb', '\u2026', '\u00a0', '\u00c0', '\u00c3',
      '\u00d5', '\u0152', '\u0153', '\u2013', '\u2014', '\u201c', '\u201d', '\u2018', '\u2019', '\u00f7',
      '\u25ca', '\u00ff', '\u0178', '\u2044', '\u20ac', '\u2039', '\u203a', '\ufb01', '\ufb02', '\u2021',
      '\u00b7', '\u201a', '\u201e', '\u2030', '\u00c2', '\u00ca', '\u00c1', '\u00cb', '\u00c8', '\u00cd',
      '\u00ce', '\u00cf', '\u00cc', '\u00d3', '\u00d4', '\uf8ff', '\u00d2', '\u00da', '\u00db', '\u00d9',
      '\u0131', '\u02c6', '\u02dc', '\u00af', '\u02d8', '\u02d9', '\u02da', '\u00b8', '\u02dd', '\u02db',
      '\u02c7' };
03032 
03033             private static string GetMacRomanString(byte[] bytes)
03034             {
03035                 StringBuilder bld = new StringBuilder(bytes.Length);
03036 
```

```
03037                 for (int i = 0; i < bytes.Length; i++)
03038                 {
03039                     if (bytes[i] >= 32)
03040                     {
03041                         bld.Append(MacRomanChars[bytes[i] - 32]);
03042                     }
03043                     else
03044                     {
03045                         bld.Append((char)bytes[i]);
03046                     }
03047                 }
03048 
03049                 return bld.ToString();
03050 
03051             }
03052         }
03053 
03054         internal class TrueTypeOS2Table :  ITrueTypeTable
03055         {
03056             public ushort Version;
03057             public short XAvgCharWidth;
03058             public ushort UsWeightClass;
03059             public ushort UsWidthClass;
03060             public short FsType;
03061             public short YSubscriptXSize;
03062             public short YSubscriptYSize;
03063             public short YSubscriptXOffset;
03064             public short YSubscriptYOffset;
03065             public short YSuperscriptXSize;
03066             public short YSuperscriptYSize;
03067             public short YSuperscriptXOffset;
03068             public short YSuperscriptYOffset;
03069             public short YStrikeoutSize;
03070             public short YStrikeoutPosition;
03071             public byte SFamilyClass;
03072             public byte SFamilySubClass;
03073             public PANOSE Panose;
03074             public uint[] UlUnicodeRange;
03075             public byte[] AchVendID;
03076             public ushort FsSelection;
03077             public ushort FsFirstCharIndex;
03078             public ushort FsLastCharIndex;
03079             public short STypoAscender;
03080             public short STypoDescender;
03081             public short STypoLineGap;
03082             public ushort UsWinAscent;
03083             public ushort UsWinDescent;
03084 
03085             public uint[] UlCodePageRange;
03086 
03087             public short SxHeight;
03088             public short SCapHeight;
03089             public ushort UsDefaultChar;
03090             public ushort UsBreakChar;
03091             public ushort UsMaxContext;
03092 
03093             public ushort UsLowerOpticalPointSize;
03094             public ushort UsUpperOpticalPointSize;
03095 
03096             public byte[] GetBytes()
03097             {
03098                 using (MemoryStream ms = new MemoryStream())
03099                 {
03100                     ms.WriteUShort(this.Version);
03101                     ms.WriteShort(this.XAvgCharWidth);
03102                     ms.WriteUShort(this.UsWeightClass);
03103                     ms.WriteUShort(this.UsWidthClass);
03104                     ms.WriteShort(this.FsType);
03105                     ms.WriteShort(this.YSubscriptXSize);
03106                     ms.WriteShort(this.YSubscriptYSize);
03107                     ms.WriteShort(this.YSubscriptXOffset);
03108                     ms.WriteShort(this.YSubscriptYOffset);
03109                     ms.WriteShort(this.YSuperscriptXSize);
03110                     ms.WriteShort(this.YSuperscriptYSize);
03111                     ms.WriteShort(this.YSuperscriptXOffset);
03112                     ms.WriteShort(this.YSuperscriptYOffset);
03113                     ms.WriteShort(this.YStrikeoutSize);
03114                     ms.WriteShort(this.YStrikeoutPosition);
03115                     ms.WriteByte(this.SFamilyClass);
03116                     ms.WriteByte(this.SFamilySubClass);
03117                     ms.Write(this.Panose.GetBytes(), 0, 10);
03118                     for (int i = 0; i < this.UlUnicodeRange.Length; i++)
03119                     {
03120                         ms.WriteUInt(this.UlUnicodeRange[i]);
03121                     }
03122                     ms.Write(this.AchVendID, 0, this.AchVendID.Length);
03123                     ms.WriteUShort(this.FsSelection);
```

```
03124                     ms.WriteUShort(this.FsFirstCharIndex);
03125                     ms.WriteUShort(this.FsLastCharIndex);
03126                     ms.WriteShort(this.STypoAscender);
03127                     ms.WriteShort(this.STypoDescender);
03128                     ms.WriteShort(this.STypoLineGap);
03129                     ms.WriteUShort(this.UsWinAscent);
03130                     ms.WriteUShort(this.UsWinDescent);
03131 
03132                     if (this.Version >= 1)
03133                     {
03134                         for (int i = 0; i < this.UlCodePageRange.Length; i++)
03135                         {
03136                             ms.WriteUInt(this.UlCodePageRange[i]);
03137                         }
03138 
03139                         if (this.Version >= 2)
03140                         {
03141                             ms.WriteShort(this.SxHeight);
03142                             ms.WriteShort(this.SCapHeight);
03143                             ms.WriteUShort(this.UsDefaultChar);
03144                             ms.WriteUShort(this.UsBreakChar);
03145                             ms.WriteUShort(this.UsMaxContext);
03146 
03147                             if (this.Version >= 5)
03148                             {
03149                                 ms.WriteUShort(this.UsLowerOpticalPointSize);
03150                                 ms.WriteUShort(this.UsUpperOpticalPointSize);
03151                             }
03152                         }
03153                     }
03154 
03155                     return ms.ToArray();
03156                 }
03157             }
03158 
03159             internal struct PANOSE
03160             {
03161                 public byte BFamilyType;
03162                 public byte BSerifStyle;
03163                 public byte BWeight;
03164                 public byte BProportion;
03165                 public byte BContrast;
03166                 public byte BStrokeVariation;
03167                 public byte BArmStyle;
03168                 public byte BLetterform;
03169                 public byte BMidline;
03170                 public byte BXHeight;
03171 
03172                 public PANOSE(byte bFamilyType, byte bSerifStyle, byte bWeight, byte bProportion, byte
      bContrast, byte bStrokeVariation, byte bArmStyle, byte bLetterform, byte bMidline, byte bXHeight)
03173                 {
03174                     this.BFamilyType = bFamilyType;
03175                     this.BSerifStyle = bSerifStyle;
03176                     this.BWeight = bWeight;
03177                     this.BProportion = bProportion;
03178                     this.BContrast = bContrast;
03179                     this.BStrokeVariation = bStrokeVariation;
03180                     this.BArmStyle = bArmStyle;
03181                     this.BLetterform = bLetterform;
03182                     this.BMidline = bMidline;
03183                     this.BXHeight = bXHeight;
03184                 }
03185 
03186                 public byte[] GetBytes()
03187                 {
03188                     return new byte[] { BFamilyType, BSerifStyle, BWeight, BProportion, BContrast,
      BStrokeVariation, BArmStyle, BLetterform, BMidline, BXHeight };
03189                 }
03190             }
03191         }
03192 
03193         internal class TrueTypePostTable :  ITrueTypeTable
03194         {
03195             public Fixed Version;
03196             public Fixed ItalicAngle;
03197             public short UnderlinePosition;
03198             public short UnderlineThickness;
03199             public uint IsFixedPitch;
03200             public uint MinMemType42;
03201             public uint MaxMemType42;
03202             public uint MinMemType1;
03203             public uint MaxMemType1;
03204 
03205             public ushort NumGlyphs;
03206             public ushort[] GlyphNameIndex;
03207             public string[] Names;
03208 
```

```
03209             public byte[] GetBytes()
03210             {
03211                 using (MemoryStream ms = new MemoryStream())
03212                 {
03213                     ms.WriteFixed(this.Version);
03214                     ms.WriteFixed(this.ItalicAngle);
03215                     ms.WriteShort(this.UnderlinePosition);
03216                     ms.WriteShort(this.UnderlineThickness);
03217                     ms.WriteUInt(this.IsFixedPitch);
03218                     ms.WriteUInt(this.MinMemType42);
03219                     ms.WriteUInt(this.MaxMemType42);
03220                     ms.WriteUInt(this.MinMemType1);
03221                     ms.WriteUInt(this.MaxMemType1);
03222 
03223                     if (this.Version.Bits == 0x00020000)
03224                     {
03225                         ms.WriteUShort(NumGlyphs);
03226                         for (int i = 0; i < GlyphNameIndex.Length; i++)
03227                         {
03228                             ms.WriteUShort(GlyphNameIndex[i]);
03229                         }
03230 
03231                         for (int i = 0; i < Names.Length; i++)
03232                         {
03233                             ms.WritePascalString(Names[i]);
03234                         }
03235                     }
03236 
03237                     return ms.ToArray();
03238                 }
03239             }
03240         }
03241 
03242         internal class TrueTypeHmtxTable :  ITrueTypeTable
03243         {
03244             public LongHorFixed[] HMetrics;
03245             public short[] LeftSideBearing;
03246 
03247             public byte[] GetBytes()
03248             {
03249                 using (MemoryStream ms = new MemoryStream())
03250                 {
03251                     for (int i = 0; i < HMetrics.Length; i++)
03252                     {
03253                         ms.WriteUShort(HMetrics[i].AdvanceWidth);
03254                         ms.WriteShort(HMetrics[i].LeftSideBearing);
03255                     }
03256                     for (int i = 0; i < LeftSideBearing.Length; i++)
03257                     {
03258                         ms.WriteShort(LeftSideBearing[i]);
03259                     }
03260 
03261                     return ms.ToArray();
03262                 }
03263             }
03264         }
03265 
03266         internal class TrueTypeMaxpTable :  ITrueTypeTable
03267         {
03268             public Fixed Version;
03269             public ushort NumGlyphs;
03270             public ushort MaxPoints;
03271             public ushort MaxContours;
03272             public ushort MaxComponentPoints;
03273             public ushort MaxComponentContours;
03274             public ushort MaxZones;
03275             public ushort MaxTwilightPoints;
03276             public ushort MaxStorage;
03277             public ushort MaxFunctionDefs;
03278             public ushort MaxInstructionDefs;
03279             public ushort MaxStackElements;
03280             public ushort MaxSizeOfInstructions;
03281             public ushort MaxComponentElements;
03282             public ushort MaxComponentDepth;
03283 
03284             public byte[] GetBytes()
03285             {
03286                 using (MemoryStream ms = new MemoryStream())
03287                 {
03288                     ms.WriteFixed(this.Version);
03289                     ms.WriteUShort(this.NumGlyphs);
03290                     ms.WriteUShort(this.MaxPoints);
03291                     ms.WriteUShort(this.MaxContours);
03292                     ms.WriteUShort(this.MaxComponentPoints);
03293                     ms.WriteUShort(this.MaxComponentContours);
03294                     ms.WriteUShort(this.MaxZones);
03295                     ms.WriteUShort(this.MaxTwilightPoints);
```

```
03296                     ms.WriteUShort(this.MaxStorage);
03297                     ms.WriteUShort(this.MaxFunctionDefs);
03298                     ms.WriteUShort(this.MaxInstructionDefs);
03299                     ms.WriteUShort(this.MaxStackElements);
03300                     ms.WriteUShort(this.MaxSizeOfInstructions);
03301                     ms.WriteUShort(this.MaxComponentElements);
03302                     ms.WriteUShort(this.MaxComponentDepth);
03303                     return ms.ToArray();
03304                 }
03305             }
03306         }
03307 
03308         internal class TrueTypeHeadTable :  ITrueTypeTable
03309         {
03310             public Fixed Version;
03311             public Fixed FontRevision;
03312             public uint ChecksumAdjustment;
03313             public uint MagicNumber;
03314             public ushort Flags;
03315             public ushort UnitsPerEm;
03316             public DateTime Created;
03317             public DateTime Modified;
03318             public short XMin;
03319             public short YMin;
03320             public short XMax;
03321             public short YMax;
03322             public ushort MacStyle;
03323             public ushort LowestRecPPEM;
03324             public short FontDirectionInt;
03325             public short IndexToLocFormat;
03326             public short GlyphDataFormat;
03327 
03328             public byte[] GetBytes()
03329             {
03330                 using (MemoryStream ms = new MemoryStream())
03331                 {
03332                     ms.WriteFixed(this.Version);
03333                     ms.WriteFixed(this.FontRevision);
03334                     ms.WriteUInt(this.ChecksumAdjustment);
03335                     ms.WriteUInt(this.MagicNumber);
03336                     ms.WriteUShort(this.Flags);
03337                     ms.WriteUShort(this.UnitsPerEm);
03338                     ms.WriteDate(this.Created);
03339                     ms.WriteDate(this.Modified);
03340                     ms.WriteShort(this.XMin);
03341                     ms.WriteShort(this.YMin);
03342                     ms.WriteShort(this.XMax);
03343                     ms.WriteShort(this.YMax);
03344                     ms.WriteUShort(this.MacStyle);
03345                     ms.WriteUShort(this.LowestRecPPEM);
03346                     ms.WriteShort(this.FontDirectionInt);
03347                     ms.WriteShort(this.IndexToLocFormat);
03348                     ms.WriteShort(this.GlyphDataFormat);
03349                     return ms.ToArray();
03350                 }
03351             }
03352 
03353         }
03354 
03355         internal class TrueTypeHHeaTable :  ITrueTypeTable
03356         {
03357             public Fixed Version;
03358             public short Ascent;
03359             public short Descent;
03360             public short LineGap;
03361             public ushort AdvanceWidthMax;
03362             public short MinLeftSideBearing;
03363             public short MinRightSideBearing;
03364             public short XMaxExtent;
03365             public short CaretSlopeRise;
03366             public short CaretSlopeRun;
03367             public short CaretOffset;
03368             public short MetricDataFormat;
03369             public ushort NumOfLongHorMetrics;
03370 
03371             public byte[] GetBytes()
03372             {
03373                 using (MemoryStream ms = new MemoryStream())
03374                 {
03375                     ms.WriteFixed(this.Version);
03376                     ms.WriteShort(this.Ascent);
03377                     ms.WriteShort(this.Descent);
03378                     ms.WriteShort(this.LineGap);
03379                     ms.WriteUShort(this.AdvanceWidthMax);
03380                     ms.WriteShort(this.MinLeftSideBearing);
03381                     ms.WriteShort(this.MinRightSideBearing);
03382                     ms.WriteShort(this.XMaxExtent);
```

```
03383                     ms.WriteShort(this.CaretSlopeRise);
03384                     ms.WriteShort(this.CaretSlopeRun);
03385                     ms.WriteShort(this.CaretOffset);
03386                     ms.WriteShort(0);
03387                     ms.WriteShort(0);
03388                     ms.WriteShort(0);
03389                     ms.WriteShort(0);
03390                     ms.WriteShort(this.MetricDataFormat);
03391                     ms.WriteUShort(this.NumOfLongHorMetrics);
03392                     return ms.ToArray();
03393                 }
03394             }
03395         }
03396 
03397 
03398         internal class LangSysTable
03399         {
03400             public string Tag;
03401             public ushort LookupOrderOffset;
03402             public ushort RequiredFeatureIndex;
03403             public ushort[] FeatureIndices;
03404 
03405             public LangSysTable(string tag, Stream fs)
03406             {
03407                 this.Tag = tag;
03408                 this.LookupOrderOffset = fs.ReadUShort();
03409                 this.RequiredFeatureIndex = fs.ReadUShort();
03410                 ushort featureIndexCount = fs.ReadUShort();
03411 
03412                 this.FeatureIndices = new ushort[featureIndexCount];
03413 
03414                 for (int i = 0; i < featureIndexCount; i++)
03415                 {
03416                     this.FeatureIndices[i] = fs.ReadUShort();
03417                 }
03418             }
03419 
03420             public byte[] GetBytes()
03421             {
03422                 using (MemoryStream ms = new MemoryStream(6 + this.FeatureIndices.Length * 2))
03423                 {
03424                     ms.WriteUShort(this.LookupOrderOffset);
03425                     ms.WriteUShort(this.RequiredFeatureIndex);
03426                     ms.WriteUShort((ushort)this.FeatureIndices.Length);
03427                     for (int i = 0; i < this.FeatureIndices.Length; i++)
03428                     {
03429                         ms.WriteUShort(this.FeatureIndices[i]);
03430                     }
03431 
03432                     return ms.ToArray();
03433                 }
03434             }
03435         }
03436 
03437         internal class ScriptTable
03438         {
03439             public string Tag;
03440             public LangSysTable DefaultLangSys;
03441             public LangSysTable[] LangSysRecords;
03442 
03443             public ScriptTable(string tag, Stream fs)
03444             {
03445                 this.Tag = tag;
03446 
03447                 long startPosition = fs.Position;
03448 
03449                 ushort defaultLangSysOffset = fs.ReadUShort();
03450                 ushort langSysCount = fs.ReadUShort();
03451 
03452                 (string, ushort)[] langSysRecords = new (string, ushort)[langSysCount];
03453 
03454                 for (int i = 0; i < langSysCount; i++)
03455                 {
03456                     StringBuilder langSysTag = new StringBuilder(4);
03457                     langSysTag.Append((char)fs.ReadByte());
03458                     langSysTag.Append((char)fs.ReadByte());
03459                     langSysTag.Append((char)fs.ReadByte());
03460                     langSysTag.Append((char)fs.ReadByte());
03461 
03462                     langSysRecords[i] = (langSysTag.ToString(), fs.ReadUShort());
03463                 }
03464 
03465                 if (defaultLangSysOffset != 0)
03466                 {
03467                     fs.Seek(startPosition + defaultLangSysOffset, SeekOrigin.Begin);
03468                     this.DefaultLangSys = new LangSysTable(null, fs);
03469                 }
```

```
03470                 else
03471                 {
03472                     this.DefaultLangSys = null;
03473                 }
03474 
03475                 this.LangSysRecords = new LangSysTable[langSysCount];
03476 
03477                 for (int i = 0; i < langSysCount; i++)
03478                 {
03479                     fs.Seek(startPosition + langSysRecords[i].Item2, SeekOrigin.Begin);
03480                     this.LangSysRecords[i] = new LangSysTable(langSysRecords[i].Item1, fs);
03481                 }
03482             }
03483 
03484             public byte[] GetBytes()
03485             {
03486                 using (MemoryStream ms = new MemoryStream())
03487                 {
03488                     int offset = 6 * this.LangSysRecords.Length + 2 + 2;
03489 
03490                     if (this.DefaultLangSys != null)
03491                     {
03492                         ms.WriteUShort((ushort)offset);
03493                     }
03494                     else
03495                     {
03496                         ms.WriteUShort(0);
03497                     }
03498 
03499                     ms.WriteUShort((ushort)this.LangSysRecords.Length);
03500 
03501                     byte[] defaultLangSysTable = DefaultLangSys != null ?  DefaultLangSys.GetBytes() :
    new byte[0];
03502 
03503                     byte[][] langSysTables = new byte[this.LangSysRecords.Length][];
03504 
03505                     offset += defaultLangSysTable.Length;
03506 
03507                     int[] offsets = new int[this.LangSysRecords.Length];
03508 
03509                     for (int i = 0; i < this.LangSysRecords.Length; i++)
03510                     {
03511                         offsets[i] = offset;
03512                         langSysTables[i] = this.LangSysRecords[i].GetBytes();
03513                         offset += langSysTables[i].Length;
03514                     }
03515 
03516                     for (int i = 0; i < this.LangSysRecords.Length; i++)
03517                     {
03518                         ms.WriteByte((byte)this.LangSysRecords[i].Tag[0]);
03519                         ms.WriteByte((byte)this.LangSysRecords[i].Tag[1]);
03520                         ms.WriteByte((byte)this.LangSysRecords[i].Tag[2]);
03521                         ms.WriteByte((byte)this.LangSysRecords[i].Tag[3]);
03522 
03523                         ms.WriteUShort((ushort)offsets[i]);
03524                     }
03525 
03526                     ms.Write(defaultLangSysTable, 0, defaultLangSysTable.Length);
03527 
03528                     for (int i = 0; i < langSysTables[i].Length; i++)
03529                     {
03530                         ms.Write(langSysTables[i], 0, langSysTables[i].Length);
03531                     }
03532 
03533                     return ms.ToArray();
03534                 }
03535             }
03536         }
03537 
03538 
03539         internal class ScriptList
03540         {
03541             public ScriptTable[] ScriptRecords;
03542             public ScriptList(Stream fs)
03543             {
03544                 long startPosition = fs.Position;
03545 
03546                 ushort scriptCount = fs.ReadUShort();
03547 
03548                 (string, ushort)[] scriptRecords = new (string, ushort)[scriptCount];
03549 
03550                 for (int i = 0; i < scriptCount; i++)
03551                 {
03552                     StringBuilder scriptTag = new StringBuilder(4);
03553                     scriptTag.Append((char)fs.ReadByte());
03554                     scriptTag.Append((char)fs.ReadByte());
03555                     scriptTag.Append((char)fs.ReadByte());
```

```
03556                   scriptTag.Append((char)fs.ReadByte());
03557
03558                   scriptRecords[i] = (scriptTag.ToString(), fs.ReadUShort());
03559               }
03560
03561               this.ScriptRecords = new ScriptTable[scriptCount];
03562
03563               for (int i = 0; i < scriptCount; i++)
03564               {
03565                   fs.Seek(startPosition + scriptRecords[i].Item2, SeekOrigin.Begin);
03566                   this.ScriptRecords[i] = new ScriptTable(scriptRecords[i].Item1, fs);
03567               }
03568           }
03569
03570           public byte[] GetBytes()
03571           {
03572               using (MemoryStream ms = new MemoryStream())
03573               {
03574                   int offset = 2 + 6 * this.ScriptRecords.Length;
03575
03576                   ms.WriteUShort((ushort)this.ScriptRecords.Length);
03577
03578                   byte[][] scripts = new byte[this.ScriptRecords.Length][];
03579                   int[] offsets = new int[this.ScriptRecords.Length];
03580
03581                   for (int i = 0; i < this.ScriptRecords.Length; i++)
03582                   {
03583                       offsets[i] = offset;
03584                       scripts[i] = this.ScriptRecords[i].GetBytes();
03585                       offset += scripts[i].Length;
03586                   }
03587
03588                   for (int i = 0; i < this.ScriptRecords.Length; i++)
03589                   {
03590                       ms.WriteByte((byte)this.ScriptRecords[i].Tag[0]);
03591                       ms.WriteByte((byte)this.ScriptRecords[i].Tag[1]);
03592                       ms.WriteByte((byte)this.ScriptRecords[i].Tag[2]);
03593                       ms.WriteByte((byte)this.ScriptRecords[i].Tag[3]);
03594
03595                       ms.WriteUShort((ushort)offsets[i]);
03596                   }
03597
03598                   for (int i = 0; i < this.ScriptRecords.Length; i++)
03599                   {
03600                       ms.Write(scripts[i], 0, scripts[i].Length);
03601                   }
03602
03603                   return ms.ToArray();
03604               }
03605           }
03606       }
03607
03608       internal class LookupTable
03609       {
03610           public ushort LookupType;
03611           public ushort LookupFlag;
03612           public ITrueTypeTable[] SubTables;
03613           public ushort MarkFilteringSet;
03614
03615           public LookupTable(Stream fs)
03616           {
03617               long startPosition = fs.Position;
03618
03619               this.LookupType = fs.ReadUShort();
03620               this.LookupFlag = fs.ReadUShort();
03621
03622               ushort subTableCount = fs.ReadUShort();
03623               ushort[] subTableOffsets = new ushort[subTableCount];
03624
03625               for (int i = 0; i < subTableCount; i++)
03626               {
03627                   subTableOffsets[i] = fs.ReadUShort();
03628               }
03629
03630               if ((this.LookupFlag & 0x0010) != 0)
03631               {
03632                   this.MarkFilteringSet = fs.ReadUShort();
03633               }
03634
03635               if (this.LookupType == 1)
03636               {
03637                   this.SubTables = new ITrueTypeTable[subTableCount];
03638
03639                   for (int i = 0; i < subTableCount; i++)
03640                   {
03641                       fs.Seek(startPosition + subTableOffsets[i], SeekOrigin.Begin);
03642                       this.SubTables[i] = new SinglePosSubtable(fs);
```

```
03643                   }
03644               }
03645               else if (this.LookupType == 2)
03646               {
03647                   this.SubTables = new ITrueTypeTable[subTableCount];
03648
03649                   for (int i = 0; i < subTableCount; i++)
03650                   {
03651                       fs.Seek(startPosition + subTableOffsets[i], SeekOrigin.Begin);
03652                       this.SubTables[i] = new PairPosSubtable(fs);
03653                   }
03654               }
03655               else if (this.LookupType == 9)
03656               {
03657                   for (int i = 0; i < subTableCount; i++)
03658                   {
03659                       fs.Seek(startPosition + subTableOffsets[i], SeekOrigin.Begin);
03660
03661                       long extensionSubtableStartPosition = fs.Position;
03662
03663                       ushort posFormat = fs.ReadUShort(); // 1
03664                       ushort extensionLookupType = fs.ReadUShort();
03665                       uint offset = fs.ReadUInt();
03666
03667                       if (i == 0)
03668                       {
03669                           this.LookupType = extensionLookupType;
03670
03671                           if (this.LookupType == 1 || this.LookupType == 2)
03672                           {
03673                               this.SubTables = new ITrueTypeTable[subTableCount];
03674                           }
03675                           else
03676                           {
03677                               this.SubTables = new ITrueTypeTable[0];
03678                           }
03679                       }
03680
03681                       if (this.LookupType == 1)
03682                       {
03683                           fs.Seek(extensionSubtableStartPosition + offset, SeekOrigin.Begin);
03684                           this.SubTables[i] = new SinglePosSubtable(fs);
03685                       }
03686                       else if (this.LookupType == 2)
03687                       {
03688                           fs.Seek(extensionSubtableStartPosition + offset, SeekOrigin.Begin);
03689                           this.SubTables[i] = new PairPosSubtable(fs);
03690                       }
03691                   }
03692               }
03693               else
03694               {
03695                   this.SubTables = new ITrueTypeTable[0];
03696               }
03697           }
03698
03699           public byte[] GetBytes()
03700           {
03701               using (MemoryStream ms = new MemoryStream())
03702               {
03703                   ms.WriteUShort(this.LookupType);
03704                   ms.WriteUShort(this.LookupFlag);
03705                   ms.WriteUShort((ushort)this.SubTables.Length);
03706
03707                   int offset = 6 + 2 * this.SubTables.Length;
03708
03709                   if ((this.LookupFlag & 0x0010) != 0)
03710                   {
03711                       offset += 2;
03712                   }
03713
03714                   byte[][] subTables = new byte[this.SubTables.Length][];
03715                   int[] offsets = new int[this.SubTables.Length];
03716
03717                   for (int i = 0; i < this.SubTables.Length; i++)
03718                   {
03719                       offsets[i] = offset;
03720                       subTables[i] = this.SubTables[i].GetBytes();
03721                       offset += subTables[i].Length;
03722                   }
03723
03724                   for (int i = 0; i < this.SubTables.Length; i++)
03725                   {
03726                       ms.WriteUShort((ushort)offsets[i]);
03727                   }
03728
03729                   if ((this.LookupFlag & 0x0010) != 0)
```

```
03730                       {
03731                           ms.WriteUShort(this.MarkFilteringSet);
03732                       }
03733
03734                       for (int i = 0; i < this.SubTables.Length; i++)
03735                       {
03736                           ms.Write(subTables[i], 0, subTables[i].Length);
03737                       }
03738
03739                       return ms.ToArray();
03740                   }
03741               }
03742           }
03743
03744           internal class LookupList
03745           {
03746               public LookupTable[] LookupTables;
03747
03748               public LookupList(Stream fs)
03749               {
03750                   long startPosition = fs.Position;
03751
03752                   ushort lookupCount = fs.ReadUShort();
03753                   ushort[] lookupOffsets = new ushort[lookupCount];
03754
03755                   for (int i = 0; i < lookupCount; i++)
03756                   {
03757                       lookupOffsets[i] = fs.ReadUShort();
03758                   }
03759
03760                   this.LookupTables = new LookupTable[lookupCount];
03761
03762                   for (int i = 0; i < lookupCount; i++)
03763                   {
03764                       fs.Seek(startPosition + lookupOffsets[i], SeekOrigin.Begin);
03765                       this.LookupTables[i] = new LookupTable(fs);
03766                   }
03767               }
03768
03769               public byte[] GetBytes()
03770               {
03771                   using (MemoryStream ms = new MemoryStream())
03772                   {
03773                       ms.WriteUShort((ushort)this.LookupTables.Length);
03774
03775                       int offset = 2 + 2 * this.LookupTables.Length;
03776
03777                       int[] offsets = new int[this.LookupTables.Length];
03778                       byte[][] lookupTables = new byte[this.LookupTables.Length][];
03779
03780                       for (int i = 0; i < this.LookupTables.Length; i++)
03781                       {
03782                           offsets[i] = offset;
03783                           lookupTables[i] = this.LookupTables[i].GetBytes();
03784                           offset += lookupTables[i].Length;
03785                       }
03786
03787                       for (int i = 0; i < this.LookupTables.Length; i++)
03788                       {
03789                           ms.WriteUShort((ushort)offsets[i]);
03790                       }
03791
03792                       for (int i = 0; i < this.LookupTables.Length; i++)
03793                       {
03794                           ms.Write(lookupTables[i], 0, lookupTables[i].Length);
03795                       }
03796
03797                       return ms.ToArray();
03798                   }
03799               }
03800           }
03801
03802           internal class CoverageTable
03803           {
03804               public struct RangeRecord
03805               {
03806                   public ushort StartGlyphID;
03807                   public ushort EndGlyphID;
03808                   public ushort StartCoverageIndex;
03809
03810                   public RangeRecord(Stream fs)
03811                   {
03812                       this.StartGlyphID = fs.ReadUShort();
03813                       this.EndGlyphID = fs.ReadUShort();
03814                       this.StartCoverageIndex = fs.ReadUShort();
03815                   }
03816               }
```

```
03817
03818             public ushort CoverageFormat;
03819
03820             public ushort[] GlyphArray;
03821             public RangeRecord[] RangeRecords;
03822
03823             public int ContainsGlyph(int glyphIndex)
03824             {
03825                 if (this.CoverageFormat == 1)
03826                 {
03827                     for (int i = 0; i < this.GlyphArray.Length; i++)
03828                     {
03829                         if (this.GlyphArray[i] == glyphIndex)
03830                         {
03831                             return i;
03832                         }
03833                     }
03834
03835                     return -1;
03836                 }
03837                 else if (this.CoverageFormat == 2)
03838                 {
03839                     for (int i = 0; i < this.RangeRecords.Length; i++)
03840                     {
03841                         if (this.RangeRecords[i].StartGlyphID <= glyphIndex &&
      this.RangeRecords[i].EndGlyphID >= glyphIndex)
03842                         {
03843                             return this.RangeRecords[i].StartCoverageIndex + (glyphIndex -
      this.RangeRecords[i].StartGlyphID);
03844                         }
03845                     }
03846
03847                     return -1;
03848                 }
03849                 else
03850                 {
03851                     return -1;
03852                 }
03853             }
03854
03855             public CoverageTable(Stream fs)
03856             {
03857                 this.CoverageFormat = fs.ReadUShort();
03858
03859                 if (this.CoverageFormat == 1)
03860                 {
03861                     ushort glyphCount = fs.ReadUShort();
03862                     this.GlyphArray = new ushort[glyphCount];
03863                     for (int i = 0; i < glyphCount; i++)
03864                     {
03865                         this.GlyphArray[i] = fs.ReadUShort();
03866                     }
03867                 }
03868                 else if (this.CoverageFormat == 2)
03869                 {
03870                     ushort rangeCount = fs.ReadUShort();
03871
03872                     this.RangeRecords = new RangeRecord[rangeCount];
03873
03874                     for (int i = 0; i < rangeCount; i++)
03875                     {
03876                         this.RangeRecords[i] = new RangeRecord(fs);
03877                     }
03878                 }
03879             }
03880
03881             public byte[] GetBytes()
03882             {
03883                 if (this.CoverageFormat == 1)
03884                 {
03885                     using (MemoryStream ms = new MemoryStream(4 + 2 * this.GlyphArray.Length))
03886                     {
03887                         ms.WriteUShort(this.CoverageFormat);
03888                         ms.WriteUShort((ushort)this.GlyphArray.Length);
03889                         for (int i = 0; i < this.GlyphArray.Length; i++)
03890                         {
03891                             ms.WriteUShort(this.GlyphArray[i]);
03892                         }
03893
03894                         return ms.ToArray();
03895                     }
03896                 }
03897                 else if (this.CoverageFormat == 2)
03898                 {
03899                     using (MemoryStream ms = new MemoryStream(4 + 6 * this.RangeRecords.Length))
03900                     {
03901                         ms.WriteUShort(this.CoverageFormat);
```

```
03902                           ms.WriteUShort((ushort)this.RangeRecords.Length);
03903                           for (int i = 0; i < this.RangeRecords.Length; i++)
03904                           {
03905                               ms.WriteUShort(this.RangeRecords[i].StartGlyphID);
03906                               ms.WriteUShort(this.RangeRecords[i].EndGlyphID);
03907                               ms.WriteUShort(this.RangeRecords[i].StartCoverageIndex);
03908                           }
03909
03910                           return ms.ToArray();
03911                       }
03912                   }
03913                   else
03914                   {
03915                       using (MemoryStream ms = new MemoryStream(4 + 2 * this.GlyphArray.Length))
03916                       {
03917                           ms.WriteUShort(this.CoverageFormat);
03918
03919                           return ms.ToArray();
03920                       }
03921                   }
03922               }
03923           }
03924
03925           internal class ClassDefinitionTable
03926           {
03927               public struct ClassRangeRecord
03928               {
03929                   public ushort StartGlyphID;
03930                   public ushort EndGlyphID;
03931                   public ushort Class;
03932
03933                   public ClassRangeRecord(Stream fs)
03934                   {
03935                       this.StartGlyphID = fs.ReadUShort();
03936                       this.EndGlyphID = fs.ReadUShort();
03937                       this.Class = fs.ReadUShort();
03938                   }
03939               }
03940
03941               public ushort ClassFormat;
03942               public ushort StartGlyphID;
03943               public ushort[] ClassValueArray;
03944
03945               public ClassRangeRecord[] ClassRangeRecords;
03946
03947               public ClassDefinitionTable(Stream fs)
03948               {
03949                   this.ClassFormat = fs.ReadUShort();
03950
03951                   if (this.ClassFormat == 1)
03952                   {
03953                       this.StartGlyphID = fs.ReadUShort();
03954                       ushort glyphCount = fs.ReadUShort();
03955
03956                       this.ClassValueArray = new ushort[glyphCount];
03957
03958                       for (int i = 0; i < this.ClassValueArray.Length; i++)
03959                       {
03960                           this.ClassValueArray[i] = fs.ReadUShort();
03961                       }
03962                   }
03963                   else if (this.ClassFormat == 2)
03964                   {
03965                       ushort classRangeCount = fs.ReadUShort();
03966
03967                       this.ClassRangeRecords = new ClassRangeRecord[classRangeCount];
03968
03969                       for (int i = 0; i < classRangeCount; i++)
03970                       {
03971                           this.ClassRangeRecords[i] = new ClassRangeRecord(fs);
03972                       }
03973                   }
03974               }
03975
03976               public int GetClass(int glyphIndex)
03977               {
03978                   if (this.ClassFormat == 1)
03979                   {
03980                       if (glyphIndex >= StartGlyphID && glyphIndex - StartGlyphID <
    this.ClassValueArray.Length)
03981                       {
03982                           return this.ClassValueArray[glyphIndex - StartGlyphID];
03983                       }
03984                       else
03985                       {
03986                           return -1;
03987                       }
```

```
03988                 }
03989                 else if (this.ClassFormat == 2)
03990                 {
03991                     for (int i = 0; i < this.ClassRangeRecords.Length; i++)
03992                     {
03993                         if (this.ClassRangeRecords[i].StartGlyphID <= glyphIndex &&
      this.ClassRangeRecords[i].EndGlyphID >= glyphIndex)
03994                         {
03995                             return this.ClassRangeRecords[i].Class;
03996                         }
03997                     }
03998                     return -1;
03999                 }
04000                 else
04001                 {
04002                     return -1;
04003                 }
04004             }
04005
04006             public byte[] GetBytes()
04007             {
04008                 if (this.ClassFormat == 1)
04009                 {
04010                     using (MemoryStream ms = new MemoryStream(6 + 2 * this.ClassValueArray.Length))
04011                     {
04012                         ms.WriteUShort(this.ClassFormat);
04013                         ms.WriteUShort(this.StartGlyphID);
04014                         ms.WriteUShort((ushort)this.ClassValueArray.Length);
04015
04016                         for (int i = 0; i < this.ClassValueArray.Length; i++)
04017                         {
04018                             ms.WriteUShort(this.ClassValueArray[i]);
04019                         }
04020
04021                         return ms.ToArray();
04022                     }
04023                 }
04024                 else if (this.ClassFormat == 2)
04025                 {
04026                     using (MemoryStream ms = new MemoryStream(4 + 6 * this.ClassRangeRecords.Length))
04027                     {
04028                         ms.WriteUShort(this.ClassFormat);
04029                         ms.WriteUShort((ushort)this.ClassRangeRecords.Length);
04030
04031                         for (int i = 0; i < this.ClassRangeRecords.Length; i++)
04032                         {
04033                             ms.WriteUShort(this.ClassRangeRecords[i].StartGlyphID);
04034                             ms.WriteUShort(this.ClassRangeRecords[i].EndGlyphID);
04035                             ms.WriteUShort(this.ClassRangeRecords[i].Class);
04036                         }
04037
04038                         return ms.ToArray();
04039                     }
04040                 }
04041                 else
04042                 {
04043                     using (MemoryStream ms = new MemoryStream(2))
04044                     {
04045                         ms.WriteUShort(this.ClassFormat);
04046
04047                         return ms.ToArray();
04048                     }
04049                 }
04050             }
04051         }
04052
04053         internal class FeatureTable
04054         {
04055             public string Tag;
04056             public byte[] FeatureParams;
04057             public ushort[] LookupListIndices;
04058
04059             public FeatureTable(string tag, Stream fs)
04060             {
04061                 this.Tag = tag;
04062
04063                 long startPosition = fs.Position;
04064
04065                 ushort featureParamsOffset = fs.ReadUShort();
04066
04067                 ushort lookupIndexCount = fs.ReadUShort();
04068                 this.LookupListIndices = new ushort[lookupIndexCount];
04069
04070                 for (int i = 0; i < lookupIndexCount; i++)
04071                 {
04072                     this.LookupListIndices[i] = fs.ReadUShort();
04073                 }
```

```
04074
04075                 if (featureParamsOffset != 0 && tag == "size")
04076                 {
04077                     fs.Seek(startPosition + featureParamsOffset, SeekOrigin.Begin);
04078
04079                     this.FeatureParams = new byte[10];
04080
04081                     for (int i = 0; i < 10; i++)
04082                     {
04083                         this.FeatureParams[i] = (byte)fs.ReadByte();
04084                     }
04085                 }
04086                 else
04087                 {
04088                     this.FeatureParams = null;
04089                 }
04090             }
04091
04092             public byte[] GetBytes()
04093             {
04094                 using (MemoryStream ms = new MemoryStream())
04095                 {
04096                     if (this.FeatureParams != null)
04097                     {
04098                         ms.WriteUShort((ushort)(4 + 2 * this.LookupListIndices.Length));
04099                     }
04100                     else
04101                     {
04102                         ms.WriteUShort(0);
04103                     }
04104
04105                     ms.WriteUShort((ushort)this.LookupListIndices.Length);
04106
04107                     for (int i = 0; i < this.LookupListIndices.Length; i++)
04108                     {
04109                         ms.WriteUShort(this.LookupListIndices[i]);
04110                     }
04111
04112                     if (this.FeatureParams != null)
04113                     {
04114                         ms.Write(this.FeatureParams, 0, this.FeatureParams.Length);
04115                     }
04116
04117                     return ms.ToArray();
04118                 }
04119             }
04120         }
04121
04122         internal class FeatureList
04123         {
04124             public FeatureTable[] FeatureRecords;
04125
04126             public FeatureList(Stream fs)
04127             {
04128                 long startPosition = fs.Position;
04129
04130                 ushort featureCount = fs.ReadUShort();
04131
04132                 (string, ushort)[] featureRecords = new (string, ushort)[featureCount];
04133
04134                 for (int i = 0; i < featureCount; i++)
04135                 {
04136                     StringBuilder featureTag = new StringBuilder(4);
04137                     featureTag.Append((char)fs.ReadByte());
04138                     featureTag.Append((char)fs.ReadByte());
04139                     featureTag.Append((char)fs.ReadByte());
04140                     featureTag.Append((char)fs.ReadByte());
04141
04142                     featureRecords[i] = (featureTag.ToString(), fs.ReadUShort());
04143                 }
04144
04145                 this.FeatureRecords = new FeatureTable[featureCount];
04146
04147                 for (int i = 0; i < featureCount; i++)
04148                 {
04149                     fs.Seek(startPosition + featureRecords[i].Item2, SeekOrigin.Begin);
04150
04151                     this.FeatureRecords[i] = new FeatureTable(featureRecords[i].Item1, fs);
04152                 }
04153             }
04154
04155             public byte[] GetBytes()
04156             {
04157                 using (MemoryStream ms = new MemoryStream())
04158                 {
04159                     int offset = 2 + 6 * this.FeatureRecords.Length;
04160
```

```
04161                   ms.WriteUShort((ushort)this.FeatureRecords.Length);
04162
04163                   byte[][] featureRecords = new byte[this.FeatureRecords.Length][];
04164                   int[] offsets = new int[this.FeatureRecords.Length];
04165
04166                   for (int i = 0; i < this.FeatureRecords.Length; i++)
04167                   {
04168                       offsets[i] = offset;
04169                       featureRecords[i] = this.FeatureRecords[i].GetBytes();
04170                       offset += featureRecords[i].Length;
04171                   }
04172
04173                   for (int i = 0; i < this.FeatureRecords.Length; i++)
04174                   {
04175                       ms.WriteByte((byte)this.FeatureRecords[i].Tag[0]);
04176                       ms.WriteByte((byte)this.FeatureRecords[i].Tag[1]);
04177                       ms.WriteByte((byte)this.FeatureRecords[i].Tag[2]);
04178                       ms.WriteByte((byte)this.FeatureRecords[i].Tag[3]);
04179
04180                       ms.WriteUShort((ushort)offsets[i]);
04181                   }
04182
04183                   for (int i = 0; i < this.FeatureRecords.Length; i++)
04184                   {
04185                       ms.Write(featureRecords[i], 0, featureRecords[i].Length);
04186                   }
04187
04188                   return ms.ToArray();
04189               }
04190           }
04191       }
04192
04193       internal class ValueRecord
04194       {
04195           public short XPlacement;
04196           public short YPlacement;
04197           public short XAdvance;
04198           public short YAdvance;
04199           public ushort ValueFormat;
04200
04201           public ValueRecord(Stream fs, ushort valueFormat)
04202           {
04203               this.ValueFormat = valueFormat;
04204
04205               if ((valueFormat & 0x0001) != 0)
04206               {
04207                   this.XPlacement = fs.ReadShort();
04208               }
04209
04210               if ((valueFormat & 0x0002) != 0)
04211               {
04212                   this.YPlacement = fs.ReadShort();
04213               }
04214
04215               if ((valueFormat & 0x0004) != 0)
04216               {
04217                   this.XAdvance = fs.ReadShort();
04218               }
04219
04220               if ((valueFormat & 0x0008) != 0)
04221               {
04222                   this.YAdvance = fs.ReadShort();
04223               }
04224
04225               if ((valueFormat & 0x0010) != 0)
04226               {
04227                   fs.ReadUShort();
04228               }
04229
04230               if ((valueFormat & 0x0020) != 0)
04231               {
04232                   fs.ReadUShort();
04233               }
04234
04235               if ((valueFormat & 0x0040) != 0)
04236               {
04237                   fs.ReadUShort();
04238               }
04239
04240               if ((valueFormat & 0x0080) != 0)
04241               {
04242                   fs.ReadUShort();
04243               }
04244           }
04245
04246           public byte[] GetBytes()
04247           {
```

```
04248                 using (MemoryStream ms = new MemoryStream())
04249                 {
04250                     if ((this.ValueFormat & 0x0001) != 0)
04251                     {
04252                         ms.WriteShort(this.XPlacement);
04253                     }
04254 
04255                     if ((this.ValueFormat & 0x0002) != 0)
04256                     {
04257                         ms.WriteShort(this.YPlacement);
04258                     }
04259 
04260                     if ((this.ValueFormat & 0x0004) != 0)
04261                     {
04262                         ms.WriteShort(this.XAdvance);
04263                     }
04264 
04265                     if ((this.ValueFormat & 0x0008) != 0)
04266                     {
04267                         ms.WriteShort(this.YAdvance);
04268                     }
04269 
04270                     if ((this.ValueFormat & 0x0010) != 0)
04271                     {
04272                         ms.WriteUShort(0);
04273                     }
04274 
04275                     if ((this.ValueFormat & 0x0020) != 0)
04276                     {
04277                         ms.WriteUShort(0);
04278                     }
04279 
04280                     if ((this.ValueFormat & 0x0040) != 0)
04281                     {
04282                         ms.WriteUShort(0);
04283                     }
04284 
04285                     if ((this.ValueFormat & 0x0080) != 0)
04286                     {
04287                         ms.WriteUShort(0);
04288                     }
04289 
04290                     return ms.ToArray();
04291                 }
04292             }
04293         }
04294 
04295         internal class SinglePosSubtable :  ITrueTypeTable
04296         {
04297             public ushort PosFormat;
04298             public CoverageTable CoverageTable;
04299             public ushort ValueFormat;
04300             public ValueRecord ValueRecord;
04301             public ValueRecord[] ValueRecords;
04302 
04303             public SinglePosSubtable(Stream fs)
04304             {
04305                 long startPosition = fs.Position;
04306 
04307                 this.PosFormat = fs.ReadUShort();
04308 
04309                 if (this.PosFormat == 1)
04310                 {
04311                     ushort coverageOffset = fs.ReadUShort();
04312                     this.ValueFormat = fs.ReadUShort();
04313                     this.ValueRecord = new ValueRecord(fs, this.ValueFormat);
04314 
04315                     fs.Seek(startPosition + coverageOffset, SeekOrigin.Begin);
04316                     this.CoverageTable = new CoverageTable(fs);
04317                 }
04318                 else if (this.PosFormat == 2)
04319                 {
04320                     ushort coverageOffset = fs.ReadUShort();
04321                     this.ValueFormat = fs.ReadUShort();
04322 
04323                     ushort valueCount = fs.ReadUShort();
04324 
04325                     this.ValueRecords = new ValueRecord[valueCount];
04326 
04327                     for (int i = 0; i < valueCount; i++)
04328                     {
04329                         this.ValueRecords[i] = new ValueRecord(fs, this.ValueFormat);
04330                     }
04331 
04332                     fs.Seek(startPosition + coverageOffset, SeekOrigin.Begin);
04333                     this.CoverageTable = new CoverageTable(fs);
04334                 }
```

```
04335             }
04336
04337             public byte[] GetBytes()
04338             {
04339                 using (MemoryStream ms = new MemoryStream())
04340                 {
04341                     ms.WriteUShort(this.PosFormat);
04342
04343                     if (this.PosFormat == 1)
04344                     {
04345                         int offset = 22;
04346
04347                         ms.WriteUShort((ushort)offset);
04348                         ms.WriteUShort(this.ValueFormat);
04349
04350                         byte[] valueBytes = this.ValueRecord.GetBytes();
04351
04352                         ms.Write(valueBytes, 0, valueBytes.Length);
04353
04354                         byte[] coverageBytes = this.CoverageTable.GetBytes();
04355
04356                         ms.Write(coverageBytes, 0, coverageBytes.Length);
04357                     }
04358                     else if (this.PosFormat == 2)
04359                     {
04360                         int offset = 8 + 16 * ValueRecords.Length;
04361
04362                         ms.WriteUShort((ushort)offset);
04363                         ms.WriteUShort(this.ValueFormat);
04364                         ms.WriteUShort((ushort)this.ValueRecords.Length);
04365
04366                         for (int i = 0; i < this.ValueRecords.Length; i++)
04367                         {
04368                             byte[] valueBytes = this.ValueRecords[i].GetBytes();
04369
04370                             ms.Write(valueBytes, 0, valueBytes.Length);
04371                         }
04372
04373                         byte[] coverageBytes = this.CoverageTable.GetBytes();
04374
04375                         ms.Write(coverageBytes, 0, coverageBytes.Length);
04376                     }
04377
04378                     return ms.ToArray();
04379                 }
04380             }
04381         }
04382
04383         internal class PairValueRecord
04384         {
04385             public ushort SecondGlyph;
04386             public ValueRecord ValueRecord1;
04387             public ValueRecord ValueRecord2;
04388
04389             public PairValueRecord(Stream fs, ushort valueFormat1, ushort valueFormat2)
04390             {
04391                 this.SecondGlyph = fs.ReadUShort();
04392                 this.ValueRecord1 = new ValueRecord(fs, valueFormat1);
04393                 this.ValueRecord2 = new ValueRecord(fs, valueFormat2);
04394             }
04395
04396             public byte[] GetBytes()
04397             {
04398                 using (MemoryStream ms = new MemoryStream())
04399                 {
04400                     ms.WriteUShort(SecondGlyph);
04401
04402                     byte[] bytes = this.ValueRecord1.GetBytes();
04403                     ms.Write(bytes, 0, bytes.Length);
04404
04405                     bytes = this.ValueRecord2.GetBytes();
04406                     ms.Write(bytes, 0, bytes.Length);
04407
04408                     return ms.ToArray();
04409                 }
04410             }
04411         }
04412
04413         internal class Class2Record
04414         {
04415             public ValueRecord ValueRecord1;
04416             public ValueRecord ValueRecord2;
04417
04418             public Class2Record(Stream fs, ushort valueFormat1, ushort valueFormat2)
04419             {
04420                 this.ValueRecord1 = new ValueRecord(fs, valueFormat1);
04421                 this.ValueRecord2 = new ValueRecord(fs, valueFormat2);
```

```
04422             }
04423
04424             public byte[] GetBytes()
04425             {
04426                 using (MemoryStream ms = new MemoryStream())
04427                 {
04428                     byte[] bytes = this.ValueRecord1.GetBytes();
04429                     ms.Write(bytes, 0, bytes.Length);
04430
04431                     bytes = this.ValueRecord2.GetBytes();
04432                     ms.Write(bytes, 0, bytes.Length);
04433
04434                     return ms.ToArray();
04435                 }
04436             }
04437         }
04438
04439 /// <summary>
04440 /// Contains information describing how the position of two glyphs in a kerning pair should be
      altered.
04441 /// </summary>
04442         public class PairKerning
04443         {
04444 /// <summary>
04445 /// This vector contains the displacement that should be applied to the first glyph of the pair.
04446 /// </summary>
04447             public Point Glyph1Placement { get; }
04448
04449 /// <summary>
04450 /// This vector describes how the advance width of the first glyph should be altered.
04451 /// </summary>
04452             public Point Glyph1Advance { get; }
04453
04454 /// <summary>
04455 /// This vector contains the displacement that should be applied to the second glyph of the pair.
04456 /// </summary>
04457             public Point Glyph2Placement { get; }
04458
04459 /// <summary>
04460 /// This vector describes how the advance width of the second glyph should be altered.
04461 /// </summary>
04462             public Point Glyph2Advance { get; }
04463
04464             internal PairKerning(Point glyph1Placement, Point glyph1Advance, Point glyph2Placement,
      Point glyph2Advance)
04465             {
04466                 this.Glyph1Placement = glyph1Placement;
04467                 this.Glyph1Advance = glyph1Advance;
04468                 this.Glyph2Placement = glyph2Placement;
04469                 this.Glyph2Advance = glyph2Advance;
04470             }
04471         }
04472
04473         internal class PairPosSubtable :  ITrueTypeTable
04474         {
04475             public ushort PosFormat;
04476             public CoverageTable CoverageTable;
04477
04478             public ushort ValueFormat1;
04479             public ushort ValueFormat2;
04480
04481             public PairValueRecord[][] PairSets;
04482
04483             public ClassDefinitionTable ClassDef1;
04484             public ClassDefinitionTable ClassDef2;
04485             public Class2Record[][] Class1Records;
04486
04487             public PairPosSubtable(Stream fs)
04488             {
04489                 long startPosition = fs.Position;
04490
04491                 this.PosFormat = fs.ReadUShort();
04492
04493                 if (this.PosFormat == 1)
04494                 {
04495                     ushort coverageOffset = fs.ReadUShort();
04496
04497                     this.ValueFormat1 = fs.ReadUShort();
04498                     this.ValueFormat2 = fs.ReadUShort();
04499
04500                     ushort pairSetCount = fs.ReadUShort();
04501
04502                     ushort[] pairSetOffsets = new ushort[pairSetCount];
04503
04504                     for (int i = 0; i < pairSetCount; i++)
04505                     {
04506                         pairSetOffsets[i] = fs.ReadUShort();
```

```
04507                     }
04508
04509                     fs.Seek(startPosition + coverageOffset, SeekOrigin.Begin);
04510                     this.CoverageTable = new CoverageTable(fs);
04511
04512                     this.PairSets = new PairValueRecord[pairSetCount][];
04513
04514                     for (int i = 0; i < pairSetCount; i++)
04515                     {
04516                         fs.Seek(startPosition + pairSetOffsets[i], SeekOrigin.Begin);
04517
04518                         ushort pairValueCount = fs.ReadUShort();
04519
04520                         this.PairSets[i] = new PairValueRecord[pairValueCount];
04521
04522                         for (int j = 0; j < pairValueCount; j++)
04523                         {
04524                             this.PairSets[i][j] = new PairValueRecord(fs, this.ValueFormat1,
      this.ValueFormat2);
04525                         }
04526                     }
04527                 }
04528                 else if (this.PosFormat == 2)
04529                 {
04530                     ushort coverageOffset = fs.ReadUShort();
04531
04532                     this.ValueFormat1 = fs.ReadUShort();
04533                     this.ValueFormat2 = fs.ReadUShort();
04534
04535                     ushort class1DefOffset = fs.ReadUShort();
04536                     ushort class2DefOffset = fs.ReadUShort();
04537
04538                     ushort class1Count = fs.ReadUShort();
04539                     ushort class2Count = fs.ReadUShort();
04540
04541                     this.Class1Records = new Class2Record[class1Count][];
04542
04543                     for (int i = 0; i < class1Count; i++)
04544                     {
04545                         this.Class1Records[i] = new Class2Record[class2Count];
04546
04547                         for (int j = 0; j < class2Count; j++)
04548                         {
04549                             this.Class1Records[i][j] = new Class2Record(fs, this.ValueFormat1,
      this.ValueFormat2);
04550                         }
04551                     }
04552
04553                     fs.Seek(startPosition + coverageOffset, SeekOrigin.Begin);
04554                     this.CoverageTable = new CoverageTable(fs);
04555
04556                     fs.Seek(startPosition + class1DefOffset, SeekOrigin.Begin);
04557                     this.ClassDef1 = new ClassDefinitionTable(fs);
04558
04559                     fs.Seek(startPosition + class2DefOffset, SeekOrigin.Begin);
04560
04561                     this.ClassDef2 = new ClassDefinitionTable(fs);
04562                 }
04563             }
04564
04565             public PairKerning GetKerning(int glyph1CoverageIndex, int glyph1Index, int glyph2Index)
04566             {
04567                 if (this.PosFormat == 1)
04568                 {
04569                     for (int i = 0; i < this.PairSets[glyph1CoverageIndex].Length; i++)
04570                     {
04571                         if (this.PairSets[glyph1CoverageIndex][i].SecondGlyph == glyph2Index)
04572                         {
04573                             return new PairKerning(new
      Point(this.PairSets[glyph1CoverageIndex][i].ValueRecord1.XPlacement,
      this.PairSets[glyph1CoverageIndex][i].ValueRecord1.YPlacement),
04574                                 new Point(this.PairSets[glyph1CoverageIndex][i].ValueRecord1.XAdvance,
      this.PairSets[glyph1CoverageIndex][i].ValueRecord1.YAdvance),
04575                                 new
      Point(this.PairSets[glyph1CoverageIndex][i].ValueRecord2.XPlacement,
      this.PairSets[glyph1CoverageIndex][i].ValueRecord2.YPlacement),
04576                                 new Point(this.PairSets[glyph1CoverageIndex][i].ValueRecord2.XAdvance,
      this.PairSets[glyph1CoverageIndex][i].ValueRecord2.YAdvance));
04577                         }
04578                     }
04579
04580                     return null;
04581                 }
04582                 else if (this.PosFormat == 2)
04583                 {
04584                     int glyph1Class = this.ClassDef1.GetClass(glyph1Index);
04585                     int glyph2Class = this.ClassDef2.GetClass(glyph2Index);
```

```
04586
04587                     if (glyph1Class < 0 || glyph2Class < 0)
04588                     {
04589                         return null;
04590                     }
04591
04592                     return new PairKerning(new
      Point(this.Class1Records[glyph1Class][glyph2Class].ValueRecord1.XPlacement,
      this.Class1Records[glyph1Class][glyph2Class].ValueRecord1.YPlacement),
04593                                            new
      Point(this.Class1Records[glyph1Class][glyph2Class].ValueRecord1.XAdvance,
      this.Class1Records[glyph1Class][glyph2Class].ValueRecord1.YAdvance),
04594                                            new
      Point(this.Class1Records[glyph1Class][glyph2Class].ValueRecord2.XPlacement,
      this.Class1Records[glyph1Class][glyph2Class].ValueRecord2.YPlacement),
04595                                            new
      Point(this.Class1Records[glyph1Class][glyph2Class].ValueRecord2.XAdvance,
      this.Class1Records[glyph1Class][glyph2Class].ValueRecord2.YAdvance));
04596                 }
04597                 else
04598                 {
04599                     return null;
04600                 }
04601             }
04602
04603             public byte[] GetBytes()
04604             {
04605                 using (MemoryStream ms = new MemoryStream())
04606                 {
04607                     ms.WriteUShort(this.PosFormat);
04608
04609                     if (this.PosFormat == 1)
04610                     {
04611                         int offset = 10 + 2 * PairSets.Length;
04612
04613                         ms.WriteUShort((ushort)offset);
04614
04615                         ms.WriteUShort((ushort)ValueFormat1);
04616                         ms.WriteUShort((ushort)ValueFormat2);
04617
04618                         ms.WriteUShort((ushort)this.PairSets.Length);
04619
04620                         byte[] coverageBytes = this.CoverageTable.GetBytes();
04621
04622                         offset += coverageBytes.Length;
04623
04624                         byte[][][] pairSets = new byte[this.PairSets.Length][][];
04625                         int[] offsets = new int[this.PairSets.Length];
04626
04627                         for (int i = 0; i < this.PairSets.Length; i++)
04628                         {
04629                             offsets[i] = offset;
04630
04631                             pairSets[i] = new byte[this.PairSets[i].Length][];
04632
04633                             for (int j = 0; j < this.PairSets[i].Length; j++)
04634                             {
04635                                 pairSets[i][j] = this.PairSets[i][j].GetBytes();
04636                                 offset += pairSets[i][j].Length;
04637                             }
04638                         }
04639
04640                         for (int i = 0; i < offsets.Length; i++)
04641                         {
04642                             ms.WriteUShort((ushort)offsets[i]);
04643                         }
04644
04645                         ms.Write(coverageBytes, 0, coverageBytes.Length);
04646
04647                         for (int i = 0; i < pairSets.Length; i++)
04648                         {
04649                             for (int j = 0; j < pairSets[i].Length; j++)
04650                             {
04651                                 ms.Write(pairSets[i][j], 0, pairSets[i][j].Length);
04652                             }
04653                         }
04654
04655                         return ms.ToArray();
04656                     }
04657                     else if (this.PosFormat == 2)
04658                     {
04659                         int offset = 16;
04660
04661                         byte[][][] class1Records = new byte[this.Class1Records.Length][][];
04662
04663                         for (int i = 0; i < class1Records.Length; i++)
04664                         {
```

```
04665                           class1Records[i] = new byte[this.Class1Records[i].Length][];
04666
04667                           for (int j = 0; j < class1Records[i].Length; j++)
04668                           {
04669                               class1Records[i][j] = this.Class1Records[i][j].GetBytes();
04670                               offset += class1Records[i][j].Length;
04671                           }
04672                       }
04673
04674                       ms.WriteUShort((ushort)offset);
04675
04676                       ms.WriteUShort(this.ValueFormat1);
04677                       ms.WriteUShort(this.ValueFormat2);
04678
04679                       byte[] coverageBytes = this.CoverageTable.GetBytes();
04680
04681                       offset += coverageBytes.Length;
04682
04683                       ms.WriteUShort((ushort)offset);
04684
04685                       byte[] classDef1Bytes = this.ClassDef1.GetBytes();
04686
04687                       offset += classDef1Bytes.Length;
04688
04689                       ms.WriteUShort((ushort)offset);
04690
04691                       ms.WriteUShort((ushort)this.Class1Records.Length);
04692                       ms.WriteUShort((ushort)this.Class1Records[0].Length);
04693
04694                       for (int i = 0; i < class1Records.Length; i++)
04695                       {
04696                           for (int j = 0; j < class1Records[i].Length; j++)
04697                           {
04698                               ms.Write(class1Records[i][j], 0, class1Records[i][j].Length);
04699                           }
04700                       }
04701
04702                       ms.Write(coverageBytes, 0, coverageBytes.Length);
04703                       ms.Write(classDef1Bytes, 0, classDef1Bytes.Length);
04704
04705                       byte[] classDef2Bytes = this.ClassDef2.GetBytes();
04706                       ms.Write(classDef2Bytes, 0, classDef2Bytes.Length);
04707
04708                       return ms.ToArray();
04709                   }
04710                   else
04711                   {
04712                       return ms.ToArray();
04713                   }
04714               }
04715           }
04716
04717       }
04718
04719       internal class TrueTypeGPOSTable :  ITrueTypeTable
04720       {
04721           public ushort MajorVersion;
04722           public ushort MinorVersion;
04723
04724           public ScriptList ScriptList;
04725           public FeatureList FeatureList;
04726           public LookupList LookupList;
04727
04728           private HashSet<int> KernLookups;
04729
04730           public TrueTypeGPOSTable(Stream fs)
04731           {
04732               long startPosition = fs.Position;
04733
04734               this.MajorVersion = fs.ReadUShort();
04735               this.MinorVersion = fs.ReadUShort();
04736
04737               ushort scriptListOffset = fs.ReadUShort();
04738               ushort featureListOffset = fs.ReadUShort();
04739               ushort lookupListOffset = fs.ReadUShort();
04740
04741               if (this.MinorVersion == 1)
04742               {
04743                   fs.ReadUInt();
04744               }
04745
04746               fs.Seek(startPosition + scriptListOffset, SeekOrigin.Begin);
04747               this.ScriptList = new ScriptList(fs);
04748
04749               fs.Seek(startPosition + featureListOffset, SeekOrigin.Begin);
04750               this.FeatureList = new FeatureList(fs);
04751
```

```
04752                 fs.Seek(startPosition + lookupListOffset, SeekOrigin.Begin);
04753                 this.LookupList = new LookupList(fs);
04754 
04755                 this.KernLookups = new HashSet<int>();
04756 
04757                 for (int i = 0; i < this.FeatureList.FeatureRecords.Length; i++)
04758                 {
04759                     if (this.FeatureList.FeatureRecords[i].Tag == "kern")
04760                     {
04761                         for (int j = 0; j <
      this.FeatureList.FeatureRecords[i].LookupListIndices.Length; j++)
04762                         {
04763                             if
      (this.LookupList.LookupTables[this.FeatureList.FeatureRecords[i].LookupListIndices[j]].LookupType ==
      2)
04764                             {
04765 
      this.KernLookups.Add(this.FeatureList.FeatureRecords[i].LookupListIndices[j]);
04766                             }
04767                         }
04768                     }
04769                 }
04770 
04771             }
04772 
04773             public byte[] GetBytes()
04774             {
04775                 using (MemoryStream ms = new MemoryStream())
04776                 {
04777                     ms.WriteUShort(this.MajorVersion);
04778                     ms.WriteUShort(this.MinorVersion);
04779 
04780                     int offset = 10;
04781 
04782                     if (this.MinorVersion == 1)
04783                     {
04784                         offset += 2;
04785                     }
04786 
04787                     ms.WriteUShort((ushort)offset);
04788 
04789                     byte[] scriptList = this.ScriptList.GetBytes();
04790                     offset += scriptList.Length;
04791 
04792                     ms.WriteUShort((ushort)offset);
04793 
04794                     byte[] featureList = this.FeatureList.GetBytes();
04795                     offset += featureList.Length;
04796 
04797                     ms.WriteUShort((ushort)offset);
04798 
04799                     if (this.MinorVersion == 1)
04800                     {
04801                         ms.WriteUInt(0);
04802                     }
04803 
04804                     ms.Write(scriptList, 0, scriptList.Length);
04805                     ms.Write(featureList, 0, featureList.Length);
04806 
04807                     byte[] lookupList = this.LookupList.GetBytes();
04808 
04809                     ms.Write(lookupList, 0, lookupList.Length);
04810 
04811                     return ms.ToArray();
04812                 }
04813             }
04814 
04815 
04816             public PairKerning GetKerning(int glyph1Index, int glyph2Index)
04817             {
04818                 foreach (int index in this.KernLookups)
04819                 {
04820                     for (int i = 0; i < this.LookupList.LookupTables[index].SubTables.Length; i++)
04821                     {
04822                         if (this.LookupList.LookupTables[index].SubTables[i] is PairPosSubtable
      pairPos)
04823                         {
04824                             int coverageIndex = pairPos.CoverageTable.ContainsGlyph(glyph1Index);
04825 
04826                             if (coverageIndex >= 0)
04827                             {
04828                                 return pairPos.GetKerning(coverageIndex, glyph1Index, glyph2Index);
04829                             }
04830                         }
04831                     }
04832                 }
04833 
```

```
04834                 return null;
04835             }
04836         }
04837 
04838         internal struct TrueTypeTableOffset
04839         {
04840             public uint Checksum;
04841             public uint Offset;
04842             public uint Length;
04843 
04844             public TrueTypeTableOffset(uint checksum, uint offset, uint length)
04845             {
04846                 this.Checksum = checksum;
04847                 this.Offset = offset;
04848                 this.Length = length;
04849             }
04850         }
04851 
04852         internal struct LongHorFixed
04853         {
04854             public ushort AdvanceWidth;
04855             public short LeftSideBearing;
04856 
04857             public LongHorFixed(ushort advanceWidth, short leftSideBearing)
04858             {
04859                 this.AdvanceWidth = advanceWidth;
04860                 this.LeftSideBearing = leftSideBearing;
04861             }
04862         }
04863 
04864         internal struct Fixed
04865         {
04866             public int Bits;
04867             public int BitShifts;
04868 
04869             public Fixed(int bits, int bitShifts)
04870             {
04871                 this.Bits = bits;
04872                 this.BitShifts = bitShifts;
04873             }
04874         }
04875     }
04876 
04877     internal static class ReadUtils
04878     {
04879         public static void WritePascalString(this Stream sr, string val)
04880         {
04881             sr.WriteByte((byte)val.Length);
04882             for (int i = 0; i < val.Length; i++)
04883             {
04884                 sr.WriteByte((byte)(int)val[i]);
04885             }
04886         }
04887 
04888         public static string ReadPascalString(this Stream sr)
04889         {
04890             byte length = (byte)sr.ReadByte();
04891             StringBuilder tbr = new StringBuilder(length);
04892             for (int i = 0; i < length; i++)
04893             {
04894                 tbr.Append((char)sr.ReadByte());
04895             }
04896             return tbr.ToString();
04897         }
04898 
04899         public static uint ReadUInt(this Stream sr)
04900         {
04901             return ((uint)sr.ReadByte() « 24) | ((uint)sr.ReadByte() « 16) | ((uint)sr.ReadByte() « 8)
      | (uint)sr.ReadByte();
04902         }
04903 
04904         public static void WriteUInt(this Stream sr, uint val)
04905         {
04906             sr.Write(new byte[] { (byte)(val » 24), (byte)((val » 16) & 255), (byte)((val » 8) & 255),
      (byte)(val & 255) }, 0, 4);
04907         }
04908 
04909         public static int ReadInt(this Stream sr)
04910         {
04911             return (sr.ReadByte() « 24) | (sr.ReadByte() « 16) | (sr.ReadByte() « 8) | sr.ReadByte();
04912         }
04913 
04914         public static void WriteInt(this Stream sr, int val)
04915         {
04916             sr.WriteUInt((uint)val);
04917         }
04918 
```

```
04919         public static ushort ReadUShort(this Stream sr)
04920         {
04921             return (ushort)(((uint)sr.ReadByte() << 8) | (uint)sr.ReadByte());
04922         }
04923 
04924         public static void WriteUShort(this Stream sr, ushort val)
04925         {
04926             sr.Write(new byte[] { (byte)(val >> 8), (byte)(val & 255) }, 0, 2);
04927         }
04928 
04929         public static short ReadShort(this Stream sr)
04930         {
04931             ushort result = sr.ReadUShort();
04932             short tbr;
04933             if ((result & 0x8000) != 0)
04934             {
04935                 tbr = (short)(result - (1 << 16));
04936             }
04937             else
04938             {
04939                 tbr = (short)result;
04940             }
04941             return tbr;
04942         }
04943 
04944         public static void WriteShort(this Stream sr, short val)
04945         {
04946             sr.WriteUShort((ushort)val);
04947         }
04948 
04949         public static string ReadString(this Stream sr, int length)
04950         {
04951             StringBuilder bld = new StringBuilder(length);
04952             for (int i = 0; i < length; i++)
04953             {
04954                 bld.Append((char)sr.ReadByte());
04955             }
04956             return bld.ToString();
04957         }
04958 
04959         public static TrueTypeFile.Fixed ReadFixed(this Stream sr)
04960         {
04961             return new TrueTypeFile.Fixed(sr.ReadInt(), 16);
04962         }
04963 
04964         public static void WriteFixed(this Stream sr, TrueTypeFile.Fixed val)
04965         {
04966             sr.WriteInt(val.Bits);
04967         }
04968 
04969         public static DateTime ReadDate(this Stream sr)
04970         {
04971             long macTime = sr.ReadUInt() * 0x100000000 + sr.ReadUInt();
04972             return new DateTime(1904, 1, 1).AddTicks(macTime * 1000 * TimeSpan.TicksPerMillisecond);
04973         }
04974 
04975         public static void WriteDate(this Stream sr, DateTime date)
04976         {
04977             long macTime = date.Subtract(new DateTime(1904, 1, 1)).Ticks / 1000 /
      TimeSpan.TicksPerMillisecond;
04978             sr.WriteUInt((uint)(macTime >> 32));
04979             sr.WriteUInt((uint)(macTime & 0xFFFFFFFF));
04980         }
04981     }
04982 
04983 }
```

Index

A
VectSharp.Colour, 115
VectSharp.SolidColourBrush, 739
a
VectSharp.Colour, 115
A1
VectSharp.Filters.ColourMatrix, 128
A2
VectSharp.Filters.ColourMatrix, 129
A3
VectSharp.Filters.ColourMatrix, 129
A4
VectSharp.Filters.ColourMatrix, 129
A5
VectSharp.Filters.ColourMatrix, 129
ActionDataPointElement
VectSharp.Plots.ActionDataPointElement, 36
ActionType
VectSharp.Canvas.RenderAction, 682
VectSharp.Canvas.SKRenderAction, 722
ActionTypes
VectSharp.Canvas.RenderAction, 680
VectSharp.Canvas.SKRenderAction, 717
Add
VectSharp.PDF.AnnotationStyleCollection, 53
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupExpressionNode, 560
VectSharp.PDF.OutlineTree, 564
VectSharp.PDF.OutlineTreeNode, 569
VectSharp.SimpleFontLibrary, 703, 704
AddElement
VectSharp.ThreeD.IScene, 410
VectSharp.ThreeD.Scene, 692
AddFrame
VectSharp.Animation, 47
AddLayer
VectSharp.Canvas.SKMultiLayerRenderCanvas, 710
AddPath
VectSharp.GraphicsPath, 340
AddPlotElement
VectSharp.Plots.Plot, 615
AddPlotElements
VectSharp.Plots.Plot, 616
AddRange
VectSharp.ThreeD.IScene, 410
VectSharp.ThreeD.Scene, 692
AddSmoothSpline
VectSharp.GraphicsPath, 340
AddText
VectSharp.GraphicsPath, 340, 342
AddTextOnPath
VectSharp.GraphicsPath, 342
AddTextUnderline
VectSharp.GraphicsPath, 343
AdvanceWidth
VectSharp.Font.DetailedFontMetrics, 216
AliceBlue
VectSharp.Colours, 142
Alignment
VectSharp.Plots.ContinuousAxisLabels, 185
VectSharp.Plots.ContinuousAxisTitle, 195
VectSharp.Plots.DataLabels< T >, 208
VectSharp.Plots.TextLabel< T >, 773
AllowPageBreak
VectSharp.Markdown.MarkdownRenderer, 501
AlphaInversion
VectSharp.Filters.ColourMatrix, 127
AmbientLightSource
VectSharp.ThreeD.AmbientLightSource, 38
AmbientReflectionCoefficient
VectSharp.ThreeD.PhongMaterial, 608
AngleAttenuationExponent
VectSharp.ThreeD.MaskedLightSource, 520
VectSharp.ThreeD.SpotlightLightSource, 745
Animation
VectSharp.Animation, 46
AnnotationStyle
VectSharp.PDF.AnnotationStyle, 51
AnnotationStyleCollection
VectSharp.PDF.AnnotationStyleCollection, 53
AntiqueWhite
VectSharp.Colours, 142
Apply
VectSharp.Filters.ColourMatrix, 122–124
Aqua
VectSharp.Colours, 142
Aquamarine
VectSharp.Colours, 143
Arc
VectSharp.GraphicsPath, 343, 344
Area
VectSharp.Plots.Area< T >, 57
AreaLightSource
VectSharp.ThreeD.AreaLightSource, 62
Arguments
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupExpre
549

ArrowSize
VectSharp.Plots.ContinuousAxis, 182
Ascent
VectSharp.Font, 243
Author
VectSharp.PDF.PDFMetadata, 598
Azure
VectSharp.Colours, 143

B
VectSharp.Colour, 116
VectSharp.SolidColourBrush, 739
B1
VectSharp.Filters.ColourMatrix, 129
B2
VectSharp.Filters.ColourMatrix, 130
B3
VectSharp.Filters.ColourMatrix, 130
B4
VectSharp.Filters.ColourMatrix, 130
B5
VectSharp.Filters.ColourMatrix, 130
Background
VectSharp.Animation, 48
VectSharp.Page, 575
BackgroundColour
VectSharp.Markdown.MarkdownRenderer, 501
Bars
VectSharp.Plots.Bars< T >, 71–74
BaseFontSize
VectSharp.Markdown.MarkdownRenderer, 501
BaseImageUri
VectSharp.Markdown.MarkdownRenderer, 502
Baseline
VectSharp.Markdown.LineEventArgs, 441
VectSharp.Markdown.LineMeasuredEventArgs, 447
VectSharp.Plots.ContinuousAxisLabels, 185
VectSharp.Plots.ContinuousAxisTitle, 195
VectSharp.Plots.DataLabels< T >, 208
VectSharp.Plots.TextLabel< T >, 773
BaseLinkUri
VectSharp.Markdown.MarkdownRenderer, 502
BeamWidthAngle
VectSharp.ThreeD.SpotlightLightSource, 746
Beige
VectSharp.Colours, 143
Bias
VectSharp.Filters.ConvolutionFilter, 199
Bisque
VectSharp.Colours, 143
Black
VectSharp.Colours, 143
BlanchedAlmond
VectSharp.Colours, 144
Block
VectSharp.Markdown.BlockRenderedEventArgs, 78
VectSharp.Markdown.BlockRenderingEventArgs, 80
BlockRendered
VectSharp.Markdown.MarkdownRenderer, 515
BlockRenderedEventArgs
VectSharp.Markdown.BlockRenderedEventArgs, 78
BlockRendering
VectSharp.Markdown.MarkdownRenderer, 515
BlockRenderingEventArgs
VectSharp.Markdown.BlockRenderingEventArgs, 80
Blue
VectSharp.Colours, 144
BlueViolet
VectSharp.Colours, 144
Bold
VectSharp.PDF.OutlineTreeNode, 571
BoldFontFamily
VectSharp.Markdown.MarkdownRenderer, 502
BoldItalicFontFamily
VectSharp.Markdown.MarkdownRenderer, 502
BoldUnderlineThickness
VectSharp.Markdown.MarkdownRenderer, 502
BorderColour
VectSharp.PDF.AnnotationStyle, 51
BorderDash
VectSharp.PDF.AnnotationStyle, 51
BorderWidth
VectSharp.PDF.AnnotationStyle, 51
Bottom
VectSharp.Font.DetailedFontMetrics, 216
VectSharp.Markdown.Margins, 480
BottomRight
VectSharp.Markdown.MarkdownContext, 490
BottomRightMargin
VectSharp.Filters.BoxBlurFilter, 83
VectSharp.Filters.ColourMatrixFilter, 135
VectSharp.Filters.CompositeLocationInvariantFilter, 176
VectSharp.Filters.ConvolutionFilter, 200
VectSharp.Filters.GaussianBlurFilter, 280
VectSharp.Filters.IFilter, 372
VectSharp.Filters.MaskFilter, 524
Bounds
VectSharp.Point, 627
Box1
VectSharp.Plots.BoxPlot, 87
Box2
VectSharp.Plots.BoxPlot, 87
BoxBlurFilter
VectSharp.Filters.BoxBlurFilter, 82
BoxPlot
VectSharp.Plots.BoxPlot, 85
BoxPresentationAttributes
VectSharp.Plots.BoxPlot, 87
BoxRadius
VectSharp.Filters.BoxBlurFilter, 83

BringToFront
VectSharp.Canvas.RenderAction, 680
Brown
VectSharp.Colours, 144
Brush
VectSharp.FormattedText, 264
Bullets
VectSharp.Markdown.MarkdownRenderer, 503
BurlyWood
VectSharp.Colours, 144

CadetBlue
VectSharp.Colours, 145
CastsShadow
VectSharp.ThreeD.AmbientLightSource, 39
VectSharp.ThreeD.AreaLightSource, 64
VectSharp.ThreeD.ILightSource, 390
VectSharp.ThreeD.MaskedLightSource, 521
VectSharp.ThreeD.ParallelLightSource, 584
VectSharp.ThreeD.PointLightSource, 636
VectSharp.ThreeD.SpotlightLightSource, 746
Catalog
VectSharp.PDF.PDFDocument, 595
CategoricalBars
VectSharp.Plots.CategoricalBars< T >, 93, 94
CategoricalCoordinateSystem1D
VectSharp.Plots.CategoricalCoordinateSystem1D< T >, 96
Center
VectSharp.ThreeD.AreaLightSource, 64
Centre
VectSharp.Plots.Pie, 612
VectSharp.RadialGradientBrush, 649
VectSharp.Rectangle, 673
CentreSymbol
VectSharp.Plots.BoxPlot, 87
CentreSymbolPresentationAttributes
VectSharp.Plots.BoxPlot, 87
Chartreuse
VectSharp.Colours, 145
Children
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeNode, 562
VectSharp.PDF.OutlineTreeNode, 571
Chocolate
VectSharp.Colours, 145
Cividis
VectSharp.Gradients, 287
CividisColouring
VectSharp.Gradients, 284
Class
VectSharp.TrueTypeFile.ClassDefinitionTable.ClassRangeRecord, 98
ClassRangeRecord
VectSharp.TrueTypeFile.ClassDefinitionTable.ClassRangeRecord, 98
Clear
VectSharp.PDF.AnnotationStyleCollection, 53
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeN 560
VectSharp.PDF.OutlineTree, 565
VectSharp.PDF.OutlineTreeNode, 569
ClearPNGCache
VectSharp.RasterImage, 659
ClipAction
VectSharp.Canvas.SKRenderAction, 718
ClipMargin
VectSharp.Canvas.SKMultiLayerRenderCanvas, 714
ClippingPath
VectSharp.Canvas.RenderAction, 682
Clockwise
VectSharp.Plots.Pie, 612
Clone
VectSharp.Markdown.MarkdownContext, 487
VectSharp.Segment, 695
Close
VectSharp.GraphicsPath, 344
ClusteredBars
VectSharp.Plots.ClusteredBars, 100, 101
CodeBlockBackgroundColour
VectSharp.Markdown.MarkdownRenderer, 503
CodeFont
VectSharp.Markdown.MarkdownRenderer, 503
CodeFontBold
VectSharp.Markdown.MarkdownRenderer, 503
CodeFontBoldItalic
VectSharp.Markdown.MarkdownRenderer, 504
CodeFontItalic
VectSharp.Markdown.MarkdownRenderer, 504
CodeInlineBackgroundColour
VectSharp.Markdown.MarkdownRenderer, 504
CodeInlineMargin
VectSharp.Markdown.MarkdownRenderer, 504
Coefficients
VectSharp.Plots.PolynomialTrendLine, 640
Colour
VectSharp.GradientStop, 293
VectSharp.Markdown.FormattedString, 260
VectSharp.Markdown.MarkdownContext, 490
VectSharp.Markdown.RectangleFragment, 677
VectSharp.Markdown.TextFragment, 770
VectSharp.Markdown.UnderlineFragment, 802
VectSharp.PDF.OutlineTreeNode, 571
VectSharp.SolidColourBrush, 739
VectSharp.ThreeD.ColourMaterial, 120
VectSharp.ThreeD.PhongMaterial, 608
Colouring
VectSharp.Plots.Function2D, 271
ColourMaterial
VectSharp.ThreeD.ColourMaterial, 119
ColourMatrix
VectSharp.Filters.ColourMatrix, 122
VectSharp.Filters.ColourMatrixFilter, 135
ColourMatrixFilter
VectSharp.Filters.ColourMatrixFilter, 134

CompositeCoordinateSystem2D
VectSharp.Plots.CompositeCoordinateSystem2D< T1, T2 >, 173
CompositeLocationInvariantFilter
VectSharp.Filters.CompositeLocationInvariantFilter, 175
CompressedFrame
VectSharp.AnimatedPNG.CompressedFrame, 178
Contains
VectSharp.PDF.AnnotationStyleCollection, 54
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeNode, 560
VectSharp.PDF.OutlineTree, 565
VectSharp.PDF.OutlineTreeNode, 569
ContainsKey
VectSharp.PDF.AnnotationStyleCollection, 54
ContainsPoint
VectSharp.GraphicsPath, 345
ContentMaxX
VectSharp.Markdown.LineEventArgs, 441
VectSharp.Markdown.LineMeasuredEventArgs, 447
ContentMaxY
VectSharp.Markdown.LineEventArgs, 441
VectSharp.Markdown.LineMeasuredEventArgs, 448
ContentMinY
VectSharp.Markdown.LineEventArgs, 441
VectSharp.Markdown.LineMeasuredEventArgs, 448
Contents
VectSharp.PDF.PDFDocument, 595
ContinuousAxis
VectSharp.Plots.ContinuousAxis, 181
ContinuousAxisLabels
VectSharp.Plots.ContinuousAxisLabels, 184
ContinuousAxisTicks
VectSharp.Plots.ContinuousAxisTicks, 189
ContinuousAxisTitle
VectSharp.Plots.ContinuousAxisTitle, 193, 194
ControlPoint1
VectSharp.SplineEasing, 742
ControlPoint2
VectSharp.SplineEasing, 742
ConvolutionFilter
VectSharp.Filters.ConvolutionFilter, 198
CoordinateFunction
VectSharp.Plots.CoordinateSystem< T >, 203
VectSharp.Plots.CoordinateSystem1D< T >, 205
Coordinates
VectSharp.Plots.CategoricalCoordinateSystem1D< T >, 97
CoordinateSystem
VectSharp.Plots.Area< T >, 59
VectSharp.Plots.Bars< T >, 75
VectSharp.Plots.BoxPlot, 88
VectSharp.Plots.ClusteredBars, 102
VectSharp.Plots.ContinuousAxis, 182
VectSharp.Plots.ContinuousAxisLabels, 185
VectSharp.Plots.ContinuousAxisTicks, 190
VectSharp.Plots.ContinuousAxisTitle, 195
VectSharp.Plots.CoordinateSystem< T >, 202
VectSharp.Plots.DataLabels< T >, 208
VectSharp.Plots.DataLine< T >, 212
VectSharp.Plots.ExponentialTrendLine, 223
VectSharp.Plots.Function2D, 271
VectSharp.Plots.Grid, 359
VectSharp.Plots.IPlotElement, 408
VectSharp.Plots.LinearTrendLine, 434
VectSharp.Plots.LogarithmicTrendLine, 470
VectSharp.Plots.MovingAverageTrendLine, 527
VectSharp.Plots.Pie, 612
VectSharp.Plots.PlotElement< T >, 620
VectSharp.Plots.PolynomialTrendLine, 640
VectSharp.Plots.PowerLawTrendLine, 645
VectSharp.Plots.ScatterPoints< T >, 689
VectSharp.Plots.StackedBars, 750
VectSharp.Plots.Swarm, 763
VectSharp.Plots.TextLabel< T >, 773
VectSharp.Plots.Violin, 807
CoordinateSystem1D
VectSharp.Plots.CoordinateSystem1D< T >, 204
CoordinateSystemX
VectSharp.Plots.CompositeCoordinateSystem2D< T1, T2 >, 173
CoordinateSystemY
VectSharp.Plots.CompositeCoordinateSystem2D< T1, T2 >, 174
CopyTo
VectSharp.PDF.AnnotationStyleCollection, 54
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeN 560
VectSharp.PDF.OutlineTree, 565
VectSharp.PDF.OutlineTreeNode, 570
CopyToIGraphicsContext
VectSharp.Graphics, 301
CopyToSKRenderContext
VectSharp.Canvas.SKRenderContextInterpreter, 729, 730
Coral
VectSharp.Colours, 145
CornflowerBlue
VectSharp.Colours, 145
Cornsilk
VectSharp.Colours, 146
Count
VectSharp.GradientStops, 296
VectSharp.PDF.AnnotationStyleCollection, 55
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeN 562
VectSharp.PDF.OutlineTree, 567
VectSharp.PDF.OutlineTreeNode, 571
VectSharp.Point, 633
Create
VectSharp.AnimatedPNG, 45
CreateCube

VectSharp.ThreeD.ObjectFactory, 533
CreateCuboid
VectSharp.ThreeD.ObjectFactory, 534
CreateFromHeadings
VectSharp.PDF.OutlineTree, 565
CreatePDFDocument
VectSharp.PDF.PDFContextInterpreter, 591
CreatePoints
VectSharp.ThreeD.ObjectFactory, 535
CreatePolygon
VectSharp.ThreeD.ObjectFactory, 535
CreatePrism
VectSharp.ThreeD.ObjectFactory, 536
CreateRectangle
VectSharp.ThreeD.ObjectFactory, 537
CreateSetOCGLink
VectSharp.PDF.OptionalContentGroups.OptionalContentGroup, 542
CreateSphere
VectSharp.ThreeD.ObjectFactory, 538
CreateTetrahedron
VectSharp.ThreeD.ObjectFactory, 538
CreateWireframe
VectSharp.ThreeD.ObjectFactory, 539
CreationDate
VectSharp.PDF.PDFMetadata, 598
CreationDateTimeZone
VectSharp.PDF.PDFMetadata, 598
Creator
VectSharp.PDF.PDFMetadata, 599
Crimson
VectSharp.Colours, 146
Crop
VectSharp.Graphics, 301
VectSharp.Page, 574, 575
CubicBezierTo
VectSharp.GraphicsPath, 345, 346
VectSharp.IGraphicsContext, 380
CurrentFrame
VectSharp.Canvas.AnimatedCanvas, 43
CurrentFrameProperty
VectSharp.Canvas.AnimatedCanvas, 41
CurrentLine
VectSharp.Markdown.MarkdownContext, 490
CurrentPage
VectSharp.Markdown.MarkdownContext, 490
Cursor
VectSharp.Markdown.MarkdownContext, 490
CustomProperties
VectSharp.PDF.PDFMetadata, 599
CutoffAngle
VectSharp.ThreeD.SpotlightLightSource, 746
Cyan
VectSharp.Colours, 146

DarkBlue
VectSharp.Colours, 146
DarkCyan
VectSharp.Colours, 146
DarkGoldenRod
VectSharp.Colours, 147
DarkGray
VectSharp.Colours, 147
DarkGreen
VectSharp.Colours, 147
DarkGrey
VectSharp.Colours, 147
DarkKhaki
VectSharp.Colours, 147
DarkMagenta
VectSharp.Colours, 148
DarkOliveGreen
VectSharp.Colours, 148
DarkOrange
VectSharp.Colours, 148
DarkOrchid
VectSharp.Colours, 148
DarkRed
VectSharp.Colours, 148
DarkSalmon
VectSharp.Colours, 149
DarkSeaGreen
VectSharp.Colours, 149
DarkSlateBlue
VectSharp.Colours, 149
DarkSlateGray
VectSharp.Colours, 149
DarkSlateGrey
VectSharp.Colours, 149
DarkTurquoise
VectSharp.Colours, 150
DarkViolet
VectSharp.Colours, 150
DashArray
VectSharp.LineDash, 438
Data
VectSharp.Plots.Area< T >, 59
VectSharp.Plots.Bars< T >, 75
VectSharp.Plots.ClusteredBars, 102
VectSharp.Plots.DataLabels< T >, 208
VectSharp.Plots.DataLine< T >, 212
VectSharp.Plots.MovingAverageTrendLine, 528
VectSharp.Plots.Pie, 612
VectSharp.Plots.ScatterPoints< T >, 689
VectSharp.Plots.StackedBars, 751
VectSharp.Plots.Swarm, 763
VectSharp.Plots.Violin, 808
DataHolder
VectSharp.RasterImage, 659
DataLabels
VectSharp.Plots.DataLabels< T >, 207
DataLine
VectSharp.Plots.DataLine< T >, 211
DataPointElement
VectSharp.Plots.ScatterPoints< T >, 689
DataPoints
VectSharp.Plots.Function2DGrid, 275

Deconstruct
VectSharp.ThreeD.LightIntensity, 413
DeepPink
VectSharp.Colours, 150
DeepSkyBlue
VectSharp.Colours, 150
Default
VectSharp.Canvas.FilterOption, 227
VectSharp.PDF.PDFContextInterpreter.FilterOption, 229
VectSharp.SVG.SVGContextInterpreter.FilterOption, 232
VectSharp.TextSpacing, 778
DefaultEncoding
VectSharp.PDF.PDFDocument, 595
DefaultExportState
VectSharp.PDF.OptionalContentGroups.OptionalContentGroup, 543
DefaultFillRule
VectSharp.Graphics, 335
DefaultFontLibrary
VectSharp.FontFamily, 250
DefaultPrintState
VectSharp.PDF.OptionalContentGroups.OptionalContentGroup, 543
DefaultState
VectSharp.PDF.OptionalContentGroups.OptionalContentGroup, 543
DefaultStyle
VectSharp.PDF.AnnotationStyleCollection, 55
DefaultViewState
VectSharp.PDF.OptionalContentGroups.OptionalContentGroup, 543
Descent
VectSharp.Font, 243
DestinationTag
VectSharp.PDF.OutlineTreeNode, 572
Destroy
VectSharp.TrueTypeFile, 784
DiffuseReflectionCoefficient
VectSharp.ThreeD.PhongMaterial, 608
DimGray
VectSharp.Colours, 150
DimGrey
VectSharp.Colours, 151
Direction
VectSharp.Plots.BoxPlot, 88
VectSharp.Plots.Swarm, 763
VectSharp.Plots.Violin, 808
VectSharp.ThreeD.AreaLightSource, 64
VectSharp.ThreeD.LightIntensity, 413
VectSharp.ThreeD.MaskedLightSource, 521
VectSharp.ThreeD.ParallelLightSource, 584
VectSharp.ThreeD.SpotlightLightSource, 746
Discretise
VectSharp.GraphicsPath, 346
DisposableIntPtr
VectSharp.DisposableIntPtr, 218
Dispose
VectSharp.AnimatedPNG.CompressedFrame, 179
VectSharp.Canvas.SKMultiLayerRenderCanvas, 710
VectSharp.Canvas.SKRenderAction, 718
VectSharp.DisposableIntPtr, 218
VectSharp.Filters.FilterWithRasterisableParameter, 234
VectSharp.RasterImage, 659
Disposed
VectSharp.Canvas.SKRenderAction, 723
Distance
VectSharp.ThreeD.MaskedLightSource, 521
DistanceAttenuationExponent
VectSharp.ThreeD.AreaLightSource, 64
VectSharp.ThreeD.MaskedLightSource, 521
VectSharp.ThreeD.PointLightSource, 636
VectSharp.ThreeD.SpotlightLightSource, 746
Document
VectSharp.Document, 219
VectSharp.MarkdownCanvas.MarkdownCanvasControl, 484
DocumentProperty
VectSharp.MarkdownCanvas.MarkdownCanvasControl, 482
DocumentSource
VectSharp.MarkdownCanvas.MarkdownCanvasControl, 484
DocumentSourceProperty
VectSharp.MarkdownCanvas.MarkdownCanvasControl, 483
DodgerBlue
VectSharp.Colours, 151
DrawFilteredGraphics
VectSharp.IGraphicsContext, 380
DrawFilteredGraphicsAction
VectSharp.Canvas.SKRenderAction, 718
DrawGraphics
VectSharp.Graphics, 302–304
DrawRasterImage
VectSharp.Graphics, 304–306
VectSharp.IGraphicsContext, 380
Duration
VectSharp.AnimatedPNG.CompressedFrame, 179
VectSharp.Animation, 48
VectSharp.Frame, 268
VectSharp.Transition, 781

Ease
VectSharp.IEasing, 370
VectSharp.LinearEasing, 427
VectSharp.SplineEasing, 741
Easings
VectSharp.Transition, 781
EllipticalArc
VectSharp.GraphicsPath, 347
EmojiUriResolver
VectSharp.Markdown.MarkdownRenderer, 504
EmojiURLMapping

VectSharp.Markdown.MarkdownRenderer, 505
EnableKerning
VectSharp.Font, 243
End
VectSharp.Markdown.UnderlineFragment, 802
EndGlyphID
VectSharp.TrueTypeFile.ClassDefinitionTable.ClassRangeRecord, 98
VectSharp.TrueTypeFile.CoverageTable.RangeRecord, 651
EndPoint
VectSharp.LinearGradientBrush, 430
VectSharp.Plots.ContinuousAxis, 182
VectSharp.Plots.ContinuousAxisLabels, 186
VectSharp.Plots.ContinuousAxisTicks, 190
VectSharp.Plots.ContinuousAxisTitle, 195
Equals
VectSharp.Colour, 106
VectSharp.TextSpacing, 777
ExcludeMetadata
VectSharp.PDF.PDFMetadata, 599
ExponentialTrendLine
VectSharp.Plots.ExponentialTrendLine, 221, 222

FamilyName
VectSharp.FontFamily, 250
FileName
VectSharp.FontFamily, 251
Fill
VectSharp.Canvas.RenderAction, 682
VectSharp.IGraphicsContext, 381
VectSharp.Plots.PlotElementPresentationAttributes, 622
FillPath
VectSharp.Graphics, 307
FillRectangle
VectSharp.Graphics, 307, 308
FillRule
VectSharp, 25
FillStyle
VectSharp.IGraphicsContext, 386
FillText
VectSharp.Graphics, 308, 310, 312, 313
VectSharp.IGraphicsContext, 381
FillTextOnPath
VectSharp.Graphics, 313
FillTextUnderline
VectSharp.Graphics, 315, 317
Filter
VectSharp.Canvas.SKRenderAction, 723
VectSharp.Filters.BoxBlurFilter, 82
VectSharp.Filters.ColourMatrixFilter, 134
VectSharp.Filters.CompositeLocationInvariantFilter, 176
VectSharp.Filters.ConvolutionFilter, 199
VectSharp.Filters.GaussianBlurFilter, 279
VectSharp.Filters.IFilterWithLocation, 373
VectSharp.Filters.ILocationInvariantFilter, 391
VectSharp.Filters.MaskFilter, 523
FilterOperations
VectSharp.Canvas.FilterOption, 226
VectSharp.PDF.PDFContextInterpreter.FilterOption, 229
VectSharp.SVG.SVGContextInterpreter.FilterOption, 231
FilterOption,
VectSharp.Canvas.FilterOption, 226
VectSharp.PDF.PDFContextInterpreter.FilterOption, 229
VectSharp.SVG.SVGContextInterpreter.FilterOption, 232
Filters
VectSharp.Filters.CompositeLocationInvariantFilter, 176
FireBrick
VectSharp.Colours, 151
Flatten
VectSharp.GraphicsPath, 347
VectSharp.Segment, 695
FlattenForOffsetAndGetTangents
VectSharp.Segment, 696
FloralWhite
VectSharp.Colours, 151
FocalPoint
VectSharp.RadialGradientBrush, 650
FolderFontLibrary
VectSharp.FolderFontLibrary, 237
FollowAxis
VectSharp.Plots.ContinuousAxisTitle, 195
FollowItalicAngle
VectSharp.Font.FontUnderline, 258
Font
VectSharp.Canvas.SKRenderAction, 723
VectSharp.Font, 240
VectSharp.FormattedText, 264
VectSharp.IGraphicsContext, 386
VectSharp.Markdown.MarkdownContext, 490
VectSharp.Markdown.TextFragment, 770
VectSharp.Plots.PlotElementPresentationAttributes, 622
FontFamily
VectSharp.Font, 243
VectSharp.FontFamily, 247
VectSharp.FontFamilyCreationException, 253
FontFamilyCreationException
VectSharp.FontFamilyCreationException, 253
FontSize
VectSharp.Font, 243
FontStream
VectSharp.TrueTypeFile, 795
ForbiddenAreasLeft
VectSharp.Markdown.MarkdownContext, 491
ForbiddenAreasRight
VectSharp.Markdown.MarkdownContext, 491
ForegroundColour
VectSharp.Markdown.MarkdownRenderer, 505
ForestGreen

VectSharp.Colours, 151
Format
VectSharp.FormattedText, 262, 263
VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter, 804
FormattedString
VectSharp.Markdown.FormattedString, 259
FormattedText
VectSharp.FormattedText, 262
Fragments
VectSharp.Markdown.Line, 414
Frame
VectSharp.Frame, 267
FrameCount
VectSharp.Canvas.AnimatedCanvas, 43
FrameCountProperty
VectSharp.Canvas.AnimatedCanvas, 42
FrameRate
VectSharp.Canvas.AnimatedCanvas, 43
FrameRateProperty
VectSharp.Canvas.AnimatedCanvas, 42
Frames
VectSharp.Animation, 49
FromCSSString
VectSharp.Colour, 106
FromFile
VectSharp.MuPDFUtils.PDFParser, 601–603
VectSharp.SVG.Parser, 585
FromHSL
VectSharp.Colour, 106
FromLab
VectSharp.Colour, 107
FromRgb
VectSharp.Colour, 107, 108
FromRgba
VectSharp.Colour, 109–111
FromStream
VectSharp.MuPDFUtils.PDFParser, 603–605
VectSharp.SVG.Parser, 586
FromString
VectSharp.SVG.Parser, 586
FromXYZ
VectSharp.Colour, 112
Fuchsia
VectSharp.Colours, 152
Function
VectSharp.Plots.Function2D, 271
Function2D
VectSharp.Plots.Function2D, 270
Function2DGrid
VectSharp.Plots.Function2DGrid, 274

G
VectSharp.Colour, 116
VectSharp.SolidColourBrush, 740
G1
VectSharp.Filters.ColourMatrix, 130
G2
VectSharp.Filters.ColourMatrix, 131
G3
VectSharp.Filters.ColourMatrix, 131
G4
VectSharp.Filters.ColourMatrix, 131
G5
VectSharp.Filters.ColourMatrix, 131
Gainsboro
VectSharp.Colours, 152
GaussianBlurFilter
VectSharp.Filters.GaussianBlurFilter, 279
Geometry
VectSharp.Canvas.RenderAction, 683
Get1000EmAscent
VectSharp.TrueTypeFile, 784
Get1000EmDescent
VectSharp.TrueTypeFile, 784
Get1000EmGlyphBearings
VectSharp.TrueTypeFile, 784
Get1000EmGlyphVerticalMetrics
VectSharp.TrueTypeFile, 785
Get1000EmGlyphWidth
VectSharp.TrueTypeFile, 785
Get1000EmKerning
VectSharp.TrueTypeFile, 786
Get1000EmUnderlineIntersections
VectSharp.TrueTypeFile, 787
Get1000EmUnderlinePosition
VectSharp.TrueTypeFile, 787
Get1000EmUnderlineThickness
VectSharp.TrueTypeFile, 787
Get1000EmWinAscent
VectSharp.TrueTypeFile, 788
Get1000EmXMax
VectSharp.TrueTypeFile, 788
Get1000EmXMin
VectSharp.TrueTypeFile, 788
Get1000EmYMax
VectSharp.TrueTypeFile, 788
Get1000EmYMin
VectSharp.TrueTypeFile, 789
GetAll< T >
VectSharp.Plots.Plot, 616
GetAround
VectSharp.Plots.IContinuousCoordinateSystem, 363
VectSharp.Plots.LinearCoordinateSystem2D, 421
VectSharp.Plots.LinLogCoordinateSystem2D, 453
VectSharp.Plots.LogarithmicCoordinateSystem2D, 463
VectSharp.Plots.LogLinCoordinateSystem2D, 475
GetAscent
VectSharp.Markdown.LineFragment, 443
VectSharp.Markdown.RectangleFragment, 676
VectSharp.Markdown.TextFragment, 768
VectSharp.Markdown.UnderlineFragment, 800
GetBaseline
VectSharp.Plots.Area< T >, 60
VectSharp.Plots.Bars< T >, 75

VectSharp.Plots.ClusteredBars, 102
VectSharp.Plots.StackedBars, 751
GetBounds
VectSharp.Graphics, 318
VectSharp.GraphicsPath, 348
GetColour
VectSharp.ThreeD.ColourMaterial, 119
VectSharp.ThreeD.IMaterial, 401
VectSharp.ThreeD.PhongMaterial, 607
GetColourAt
VectSharp.GradientStops, 295
GetEnumerator
VectSharp.GradientStops, 295
VectSharp.PDF.AnnotationStyleCollection, 54
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupCollection, 560
VectSharp.PDF.OutlineTree, 566
VectSharp.PDF.OutlineTreeNode, 570
VectSharp.Point, 627
GetFigures
VectSharp.GraphicsPath, 348
GetFirst< T >
VectSharp.Plots.Plot, 617
GetFirstCharIndex
VectSharp.TrueTypeFile, 789
GetFontFamilyName
VectSharp.TrueTypeFile, 789
GetFontName
VectSharp.TrueTypeFile, 789
GetFrameAtAbsolute
VectSharp.Animation, 47
GetFrameAtRelative
VectSharp.Animation, 48
GetFullFontFamilyName
VectSharp.TrueTypeFile, 790
GetGlyphIndex
VectSharp.TrueTypeFile, 790
GetGlyphPath
VectSharp.TrueTypeFile, 790, 791
GetHashCode
VectSharp.Colour, 112
VectSharp.TextSpacing, 777
GetItalicAngle
VectSharp.TrueTypeFile, 791
GetLastCharIndex
VectSharp.TrueTypeFile, 791
GetLightAt
VectSharp.ThreeD.AmbientLightSource, 39
VectSharp.ThreeD.AreaLightSource, 63
VectSharp.ThreeD.ILightSource, 389
VectSharp.ThreeD.MaskedLightSource, 520
VectSharp.ThreeD.ParallelLightSource, 583
VectSharp.ThreeD.PointLightSource, 635
VectSharp.ThreeD.SpotlightLightSource, 745
GetLinearisationPointsNormals
VectSharp.GraphicsPath, 348
GetLinearisationTangents
VectSharp.Segment, 696
GetMaxX
VectSharp.Markdown.LineFragment, 444
VectSharp.Markdown.MarkdownContext, 487, 488
VectSharp.Markdown.RectangleFragment, 676
VectSharp.Markdown.TextFragment, 768
VectSharp.Markdown.UnderlineFragment, 801
GetMaxY
VectSharp.Markdown.LineFragment, 444
VectSharp.Markdown.RectangleFragment, 676
VectSharp.Markdown.TextFragment, 768
VectSharp.Markdown.UnderlineFragment, 801
GetMinX
VectSharp.Markdown.MarkdownContext, 488, 489
GetNames
VectSharp.TrueTypeFile, 792
GetNormalAtAbsolute
VectSharp.GraphicsPath, 349
GetNormalAtRelative
VectSharp.GraphicsPath, 349
GetObstruction
VectSharp.ThreeD.AmbientLightSource, 39
VectSharp.ThreeD.AreaLightSource, 63
VectSharp.ThreeD.ILightSource, 389
VectSharp.ThreeD.MaskedLightSource, 520
VectSharp.ThreeD.ParallelLightSource, 583
VectSharp.ThreeD.PointLightSource, 635
VectSharp.ThreeD.SpotlightLightSource, 745
GetPointAt
VectSharp.Segment, 696
GetPointAtAbsolute
VectSharp.GraphicsPath, 349
GetPointAtRelative
VectSharp.GraphicsPath, 350
GetPoints
VectSharp.GraphicsPath, 350
GetStroke
VectSharp.GraphicsPath, 350
GetSyntaxHighlightedLines
VectSharp.Markdown.SyntaxHighlighter, 765
GetTags
VectSharp.Graphics, 318
GetTangentAt
VectSharp.Segment, 697
GetTangentAtAbsolute
VectSharp.GraphicsPath, 351
GetTangentAtRelative
VectSharp.GraphicsPath, 351
GetText
VectSharp.FormattedTextExtensions, 266
GetUnitsPerEm
VectSharp.TrueTypeFile, 793
GhostWhite
VectSharp.Colours, 152
Glyph1Advance
VectSharp.TrueTypeFile.PairKerning, 580
Glyph1Placement
VectSharp.TrueTypeFile.PairKerning, 580
Glyph2Advance

VectSharp.TrueTypeFile.PairKerning, 581
Glyph2Placement
VectSharp.TrueTypeFile.PairKerning, 581
Gold
VectSharp.Colours, 152
GoldenRod
VectSharp.Colours, 152
GradientStop
VectSharp.GradientStop, 292
GradientStops
VectSharp.GradientBrush, 281
VectSharp.GradientStops, 294, 295
Graphics
VectSharp.Canvas.SKRenderAction, 723
VectSharp.Frame, 268
VectSharp.Markdown.BlockRenderedEventArgs, 78
VectSharp.Markdown.BlockRenderingEventArgs, 80
VectSharp.Markdown.InlineRenderedEventArgs, 403
VectSharp.Markdown.InlineRenderingEventArgs, 405
VectSharp.Markdown.LineEventArgs, 441
VectSharp.Markdown.LineMeasuredEventArgs, 448
VectSharp.Markdown.PageFinishedEventArgs, 577
VectSharp.Markdown.PageStartedEventArgs, 579
VectSharp.Page, 575
VectSharp.Plots.GraphicsDataPointElement, 337
GraphicsDataPointElement
VectSharp.Plots.GraphicsDataPointElement, 336
Gray
VectSharp.Colours, 153
Green
VectSharp.Colours, 153
GreenYellow
VectSharp.Colours, 153
Grey
VectSharp.Colours, 153
GreyScale
VectSharp.Filters.ColourMatrix, 127
Grid
VectSharp.Plots.Grid, 358
GridType
VectSharp.Plots.Function2DGrid, 273
Groups
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupSettings, 554

H
VectSharp.Colour, 116
HasAlpha
VectSharp.RasterImage, 659
HeaderFontSizeMultipliers
VectSharp.Markdown.MarkdownRenderer, 505
HeaderLineColour
VectSharp.Markdown.MarkdownRenderer, 505
HeaderLineThicknesses
VectSharp.Markdown.MarkdownRenderer, 506
Headings
VectSharp.Markdown.MarkdownContext, 491
Height
VectSharp.Animation, 49
VectSharp.Font.DetailedFontMetrics, 216
VectSharp.IGraphicsContext, 386
VectSharp.Page, 575
VectSharp.RasterImage, 659
VectSharp.Size, 706
HoneyDew
VectSharp.Colours, 153
HotPink
VectSharp.Colours, 154

Id
VectSharp.RasterImage, 660
Identity
VectSharp.Filters.ColourMatrix, 127
ImageAction
VectSharp.Canvas.RenderAction, 680
VectSharp.Canvas.SKRenderAction, 718
ImageDataAddress
VectSharp.RasterImage, 660
ImageDestination
VectSharp.Canvas.RenderAction, 683
VectSharp.Canvas.SKRenderAction, 723
ImageId
VectSharp.Canvas.RenderAction, 683
VectSharp.Canvas.SKRenderAction, 724
ImageMarginTolerance
VectSharp.Markdown.MarkdownRenderer, 506
ImageMultiplier
VectSharp.Markdown.MarkdownRenderer, 506
ImageSideMargin
VectSharp.Markdown.MarkdownRenderer, 506
ImageSource
VectSharp.Canvas.RenderAction, 683
VectSharp.Canvas.SKRenderAction, 724
ImageUnitMultiplier
VectSharp.Markdown.MarkdownRenderer, 506
ImageUriResolver
VectSharp.Markdown.MarkdownRenderer, 507
IndentWidth
VectSharp.Markdown.MarkdownRenderer, 507
IndexOf
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeN
561
VectSharp.PDF.OutlineTree, 566
VectSharp.PDF.OutlineTreeNode, 570
IndianRed
VectSharp.Colours, 154
Indigo
VectSharp.Colours, 154
Inferno
VectSharp.Gradients, 287
InfernoColouring
VectSharp.Gradients, 284

Info
VectSharp.PDF.PDFDocument, 596
InitialAscent
VectSharp.Markdown.Line, 415
Inline
VectSharp.Markdown.InlineRenderedEventArgs, 403
VectSharp.Markdown.InlineRenderingEventArgs, 405
InlineRendered
VectSharp.Markdown.MarkdownRenderer, 515
InlineRenderedEventArgs
VectSharp.Markdown.InlineRenderedEventArgs, 403
InlineRendering
VectSharp.Markdown.MarkdownRenderer, 515
InlineRenderingEventArgs
VectSharp.Markdown.InlineRenderingEventArgs, 405
InnerRadius
VectSharp.Plots.Pie, 613
Insert
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupElement, 561
VectSharp.PDF.OutlineTree, 566
VectSharp.PDF.OutlineTreeNode, 570
InsertedColour
VectSharp.Markdown.MarkdownRenderer, 507
InsertLayer
VectSharp.Canvas.SKMultiLayerRenderCanvas, 710
Intensity
VectSharp.ThreeD.AmbientLightSource, 40
VectSharp.ThreeD.AreaLightSource, 64
VectSharp.ThreeD.LightIntensity, 413
VectSharp.ThreeD.MaskedLightSource, 521
VectSharp.ThreeD.ParallelLightSource, 584
VectSharp.ThreeD.PointLightSource, 636
VectSharp.ThreeD.SpotlightLightSource, 747
Intent
VectSharp.PDF.OptionalContentGroups.OptionalContentGroup, 543
Intents
VectSharp.PDF.OptionalContentGroups.OptionalContentConfiguration, 541
Intercept
VectSharp.Plots.ExponentialTrendLine, 223
VectSharp.Plots.LinearTrendLine, 434
VectSharp.Plots.LogarithmicTrendLine, 470
VectSharp.Plots.PowerLawTrendLine, 645
InterClusterMargin
VectSharp.Plots.ClusteredBars, 103
InterframeCompression
VectSharp.AnimatedPNG, 44
InternalAnchors
VectSharp.Markdown.MarkdownContext, 491
InternalPointer
VectSharp.DisposableIntPtr, 219
Interpolate
VectSharp.RasterImage, 660
Intersection
VectSharp.Rectangle, 670
IntervalCount
VectSharp.Plots.ContinuousAxisLabels, 186
VectSharp.Plots.ContinuousAxisTicks, 190
VectSharp.Plots.Grid, 359
IntraClusterMargin
VectSharp.Plots.ClusteredBars, 103
InvalidateAll
VectSharp.Canvas.SKRenderAction, 719
InvalidateDirty
VectSharp.Canvas.SKMultiLayerRenderCanvas, 710
InvalidateHitTestPath
VectSharp.Canvas.SKRenderAction, 719
InvalidateVisual
VectSharp.Canvas.SKRenderAction, 719
InvalidateZIndex
VectSharp.Canvas.SKMultiLayerRenderCanvas, 711
VectSharp.Canvas.SKRenderAction, 720
InverseTransform
VectSharp.Canvas.RenderAction, 683
Inversion
VectSharp.Filters.ColourMatrix, 128
InvertedAlphaShift
VectSharp.Filters.ColourMatrix, 128
IsBold
VectSharp.FontFamily, 251
VectSharp.Markdown.FormattedString, 260
VectSharp.TrueTypeFile, 793
IsDirectionStraight
VectSharp.Plots.IContinuousCoordinateSystem, 363
VectSharp.Plots.LinearCoordinateSystem2D, 421
VectSharp.Plots.LinLogCoordinateSystem2D, 453
VectSharp.Plots.LogarithmicCoordinateSystem2D, 463
VectSharp.Plots.LogLinCoordinateSystem2D, 475
IsEqual
VectSharp.Point, 628
IsFixedPitch
VectSharp.TrueTypeFile, 793
IsItalic
VectSharp.FontFamily, 251
VectSharp.Markdown.FormattedString, 260
VectSharp.TrueTypeFile, 793
IsLinear
VectSharp.Plots.IContinuousCoordinateSystem, 364
VectSharp.Plots.LinearCoordinateSystem2D, 424
VectSharp.Plots.LinLogCoordinateSystem2D, 455
VectSharp.Plots.LogarithmicCoordinateSystem2D, 465
VectSharp.Plots.LogLinCoordinateSystem2D, 477
IsNaN

VectSharp.Rectangle, 671
IsOblique
VectSharp.FontFamily, 251
VectSharp.TrueTypeFile, 794
IsOnCurve
VectSharp.TrueTypeFile.TrueTypePoint, 797
IsPlaying
VectSharp.Canvas.AnimatedCanvas, 43
IsPlayingProperty
VectSharp.Canvas.AnimatedCanvas, 42
IsReadOnly
VectSharp.PDF.AnnotationStyleCollection, 55
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeNode, 563
VectSharp.PDF.OutlineTree, 567
VectSharp.PDF.OutlineTreeNode, 572
IsScript
VectSharp.TrueTypeFile, 794
IsSerif
VectSharp.TrueTypeFile, 794
IsStandardFamily
VectSharp.FontFamily, 251
Italic
VectSharp.PDF.OutlineTreeNode, 572
ItalicFontFamily
VectSharp.Markdown.MarkdownRenderer, 507
Ivory
VectSharp.Colours, 154

Kernel
VectSharp.Filters.ConvolutionFilter, 200
Keys
VectSharp.PDF.AnnotationStyleCollection, 55
Keywords
VectSharp.PDF.PDFMetadata, 599
Khaki
VectSharp.Colours, 154

L
VectSharp.Colour, 116
Label
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeNode, 563
VectSharp.Plots.DataLabels< T >, 209
VectSharp.Plots.TextLabel< T >, 773
LabelOptionalContentGroup
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeLabel, 557
LabelString
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeLabel, 558
LabelType
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeLabel, 558
Lavender
VectSharp.Colours, 155
LavenderBlush
VectSharp.Colours, 155
LawnGreen
VectSharp.Colours, 155
LayerTransforms
VectSharp.Canvas.SKMultiLayerRenderCanvas, 714
Layout
VectSharp.Canvas.RenderAction, 684
Left
VectSharp.Markdown.Margins, 480
LeftSideBearing
VectSharp.Font.DetailedFontMetrics, 216
VectSharp.TrueTypeFile.Bearings, 76
LemonChiffon
VectSharp.Colours, 155
Library
VectSharp.Fonts.Nimbus, 532
LightBlue
VectSharp.Colours, 155
LightCoral
VectSharp.Colours, 156
LightCyan
VectSharp.Colours, 156
LightGoldenRodYellow
VectSharp.Colours, 156
LightGray
VectSharp.Colours, 156
LightGreen
VectSharp.Colours, 156
LightGrey
VectSharp.Colours, 157
LightIntensity
VectSharp.ThreeD.LightIntensity, 412
LightPink
VectSharp.Colours, 157
LightSalmon
VectSharp.Colours, 157
LightSeaGreen
VectSharp.Colours, 157
LightSkyBlue
VectSharp.Colours, 157
LightSlateGray
VectSharp.Colours, 158
LightSlateGrey
VectSharp.Colours, 158
LightSteelBlue
VectSharp.Colours, 158
LightYellow
VectSharp.Colours, 158
Lime
VectSharp.Colours, 158
LimeGreen
VectSharp.Colours, 159
Line
VectSharp.Markdown.Line, 414
VectSharp.Markdown.LineEventArgs, 441
VectSharp.Markdown.LineMeasuredEventArgs, 448
LinearCoordinateSystem1D
VectSharp.Plots.LinearCoordinateSystem1D, 416

LinearCoordinateSystem2D
VectSharp.Plots.LinearCoordinateSystem2D, 420, 421
LinearEasing
VectSharp.LinearEasing, 426
LinearGradientBrush
VectSharp.LinearGradientBrush, 428, 429
LinearisationResolution
VectSharp.Animation, 49
Linearise
VectSharp.Graphics, 318
VectSharp.GraphicsPath, 352
VectSharp.Segment, 697
LinearTrendLine
VectSharp.Plots.LinearTrendLine, 432, 433
LineCap
VectSharp.Font.FontUnderline, 258
VectSharp.IGraphicsContext, 386
VectSharp.Plots.PlotElementPresentationAttributes, 623
LineCaps
VectSharp, 25
LineDash
VectSharp.LineDash, 437
VectSharp.Plots.PlotElementPresentationAttributes, 623
LineEventArgs
VectSharp.Markdown.LineEventArgs, 440
LineJoin
VectSharp.IGraphicsContext, 387
VectSharp.Plots.PlotElementPresentationAttributes, 623
LineJoins
VectSharp, 26
LineMeasured
VectSharp.Markdown.MarkdownRenderer, 516
LineMeasuredEventArgs
VectSharp.Markdown.LineMeasuredEventArgs, 447
Linen
VectSharp.Colours, 159
LineRendered
VectSharp.Markdown.MarkdownRenderer, 516
LineRendering
VectSharp.Markdown.MarkdownRenderer, 516
LineTo
VectSharp.GraphicsPath, 352, 353
VectSharp.IGraphicsContext, 382
LineWidth
VectSharp.IGraphicsContext, 387
VectSharp.Plots.PlotElementPresentationAttributes, 623
LinkColour
VectSharp.Markdown.MarkdownRenderer, 508
LinkDestinations
VectSharp.Markdown.MarkdownContext, 492
LinkUriResolver
VectSharp.Markdown.MarkdownRenderer, 508
LinLogCoordinateSystem2D
VectSharp.Plots.LinLogCoordinateSystem2D, 451, 452
ListDepth
VectSharp.Markdown.MarkdownContext, 492
Location
VectSharp.Rectangle, 672
LogarithmicCoordinateSystem1D
VectSharp.Plots.LogarithmicCoordinateSystem1D, 457, 458
LogarithmicCoordinateSystem2D
VectSharp.Plots.LogarithmicCoordinateSystem2D, 461, 462
LogarithmicTrendLine
VectSharp.Plots.LogarithmicTrendLine, 468, 469
LogDownloads
VectSharp.Markdown.HTTPUtils, 362
LogLinCoordinateSystem2D
VectSharp.Plots.LogLinCoordinateSystem2D, 474, 475
LuminanceToAlpha
VectSharp.Filters.ColourMatrix, 125
LuminanceToColour
VectSharp.Filters.ColourMatrix, 125

Magenta
VectSharp.Colours, 159
Magma
VectSharp.Gradients, 287
MagmaColouring
VectSharp.Gradients, 285
Mako
VectSharp.Gradients, 288
MakoColouring
VectSharp.Gradients, 285
Margin
VectSharp.Plots.Bars< T >, 75
VectSharp.Plots.DataLabels< T >, 209
VectSharp.Plots.StackedBars, 751
MarginBottomRight
VectSharp.Markdown.MarkdownContext, 492
Margins
VectSharp.Markdown.Margins, 479
VectSharp.Markdown.MarkdownRenderer, 508
MarkdownCanvasControl
VectSharp.MarkdownCanvas.MarkdownCanvasControl, 482
MarkdownContext
VectSharp.Markdown.BlockRenderedEventArgs, 79
VectSharp.Markdown.BlockRenderingEventArgs, 81
VectSharp.Markdown.InlineRenderedEventArgs, 403
VectSharp.Markdown.InlineRenderingEventArgs, 405
VectSharp.Markdown.LineEventArgs, 442
VectSharp.Markdown.LineMeasuredEventArgs, 448

VectSharp.Markdown.PageFinishedEventArgs, 577
VectSharp.Markdown.PageStartedEventArgs, 579
MarkdownPipelineBuilder
VectSharp.Markdown.MarkdownRenderer, 508
MarkdownRenderer
VectSharp.Markdown.MarkdownRenderer, 499
MarkedColour
VectSharp.Markdown.MarkdownRenderer, 508
Maroon
VectSharp.Colours, 159
Mask
VectSharp.Filters.MaskFilter, 524
MaskedLightSource
VectSharp.ThreeD.MaskedLightSource, 518, 519
MaskFilter
VectSharp.Filters.MaskFilter, 523
MathFontScalingFactor
VectSharp.Markdown.MarkdownRenderer, 509
Max
VectSharp.Plots.LinearCoordinateSystem1D, 417
VectSharp.Plots.LogarithmicCoordinateSystem1D, 458
VectSharp.Point, 628
MaxBins
VectSharp.Plots.Violin, 808
MaxRenderWidth
VectSharp.MarkdownCanvas.MarkdownCanvasControl, 484
MaxRenderWidthProperty
VectSharp.MarkdownCanvas.MarkdownCanvasControl, 483
MaxX
VectSharp.Plots.ExponentialTrendLine, 223
VectSharp.Plots.Function2DGrid, 275
VectSharp.Plots.LinearCoordinateSystem2D, 424
VectSharp.Plots.LinearTrendLine, 434
VectSharp.Plots.LinLogCoordinateSystem2D, 455
VectSharp.Plots.LogarithmicCoordinateSystem2D, 465
VectSharp.Plots.LogarithmicTrendLine, 470
VectSharp.Plots.LogLinCoordinateSystem2D, 477
VectSharp.Plots.PolynomialTrendLine, 640
VectSharp.Plots.PowerLawTrendLine, 645
MaxY
VectSharp.Plots.ExponentialTrendLine, 224
VectSharp.Plots.Function2DGrid, 276
VectSharp.Plots.LinearCoordinateSystem2D, 424
VectSharp.Plots.LinearTrendLine, 434
VectSharp.Plots.LinLogCoordinateSystem2D, 455
VectSharp.Plots.LogarithmicCoordinateSystem2D, 465
VectSharp.Plots.LogarithmicTrendLine, 470
VectSharp.Plots.LogLinCoordinateSystem2D, 477
VectSharp.Plots.PolynomialTrendLine, 641
VectSharp.Plots.PowerLawTrendLine, 645
MaxZ
VectSharp.Plots.Function2DGrid, 276
Measure
VectSharp.FormattedTextExtensions, 266
VectSharp.Segment, 698
MeasureLength
VectSharp.GraphicsPath, 353
MeasureText
VectSharp.Font, 242
VectSharp.Graphics, 319
MeasureTextAdvanced
VectSharp.Font, 242
MediumAquaMarine
VectSharp.Colours, 159
MediumBlue
VectSharp.Colours, 160
MediumOrchid
VectSharp.Colours, 160
MediumPurple
VectSharp.Colours, 160
MediumSeaGreen
VectSharp.Colours, 160
MediumSlateBlue
VectSharp.Colours, 160
MediumSpringGreen
VectSharp.Colours, 161
MediumTurquoise
VectSharp.Colours, 161
MediumVioletRed
VectSharp.Colours, 161
MidnightBlue
VectSharp.Colours, 161
Min
VectSharp.Plots.LinearCoordinateSystem1D, 417
VectSharp.Plots.LogarithmicCoordinateSystem1D, 459
VectSharp.Point, 628
MinRenderWidth
VectSharp.MarkdownCanvas.MarkdownCanvasControl, 485
MinRenderWidthProperty
VectSharp.MarkdownCanvas.MarkdownCanvasControl, 483
MintCream
VectSharp.Colours, 161
MinVariation
VectSharp.MarkdownCanvas.MarkdownCanvasControl, 485
MinVariationProperty
VectSharp.MarkdownCanvas.MarkdownCanvasControl, 483
MinX
VectSharp.Plots.ExponentialTrendLine, 224
VectSharp.Plots.Function2DGrid, 276
VectSharp.Plots.LinearCoordinateSystem2D, 424
VectSharp.Plots.LinearTrendLine, 435
VectSharp.Plots.LinLogCoordinateSystem2D, 455
VectSharp.Plots.LogarithmicCoordinateSystem2D, 465
VectSharp.Plots.LogarithmicTrendLine, 471

VectSharp.Plots.LogLinCoordinateSystem2D, 477
VectSharp.Plots.PolynomialTrendLine, 641
VectSharp.Plots.PowerLawTrendLine, 646
MinY
VectSharp.Plots.ExponentialTrendLine, 224
VectSharp.Plots.Function2DGrid, 276
VectSharp.Plots.LinearCoordinateSystem2D, 425
VectSharp.Plots.LinearTrendLine, 435
VectSharp.Plots.LinLogCoordinateSystem2D, 456
VectSharp.Plots.LogarithmicCoordinateSystem2D, 466
VectSharp.Plots.LogarithmicTrendLine, 471
VectSharp.Plots.LogLinCoordinateSystem2D, 478
VectSharp.Plots.PolynomialTrendLine, 641
VectSharp.Plots.PowerLawTrendLine, 646
MinZ
VectSharp.Plots.Function2DGrid, 276
MistyRose
VectSharp.Colours, 162
Moccasin
VectSharp.Colours, 162
ModificationDate
VectSharp.PDF.PDFMetadata, 599
ModificationDateTimeZone
VectSharp.PDF.PDFMetadata, 600
Modulus
VectSharp.Point, 629
MoveLayer
VectSharp.Canvas.SKMultiLayerRenderCanvas, 711
MoveTo
VectSharp.GraphicsPath, 353, 354
VectSharp.IGraphicsContext, 382
MovingAverageTrendLine
VectSharp.Plots.MovingAverageTrendLine, 526, 527
MultiFontLibrary
VectSharp.MultiFontLibrary, 530
MultiplyOpacity
VectSharp.Brush, 91
VectSharp.GradientStop, 292
VectSharp.LinearGradientBrush, 429
VectSharp.RadialGradientBrush, 649
VectSharp.SolidColourBrush, 738
MutedRainbow
VectSharp.Gradients, 288
MutedRainbowDiscrete
VectSharp.Gradients, 288

Name
VectSharp.PDF.OptionalContentGroups.OptionalContentGroup, 544
VectSharp.TrueTypeFile.TrueTypeName, 796
NameId
VectSharp.TrueTypeFile.TrueTypeName, 796
NameIdentifier
VectSharp.TrueTypeFile.TrueTypeName, 796
NaN
VectSharp.Rectangle, 673
NavajoWhite
VectSharp.Colours, 162
Navy
VectSharp.Colours, 162
NoMetadata
VectSharp.PDF.PDFMetadata, 598
NonWhitespaceAdd
VectSharp.TextSpacing, 778
NonWhitespaceScale
VectSharp.TextSpacing, 778
Normalisation
VectSharp.Filters.ConvolutionFilter, 200
NormalisationMode
VectSharp.Plots.Plot, 615
Normalize
VectSharp.Point, 629
NotchSize
VectSharp.Plots.BoxPlot, 88
NotchWidth
VectSharp.Plots.BoxPlot, 88

Offset
VectSharp.GradientStop, 293
OkabeItoRainbow
VectSharp.Gradients, 288
OkabeItoRainbowDiscrete
VectSharp.Gradients, 289
OldLace
VectSharp.Colours, 162
Olive
VectSharp.Colours, 163
OliveDrab
VectSharp.Colours, 163
Operation
VectSharp.Canvas.FilterOption, 227
VectSharp.PDF.OptionalContentGroups.OptionalContentGroup, 544
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupAndEx 545
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupExpre 549
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupNotEx 551
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupOrExp 553
VectSharp.PDF.PDFContextInterpreter.FilterOption, 230
VectSharp.SVG.SVGContextInterpreter.FilterOption, 232
operator
VectSharp.Point, 629
operator Brush
VectSharp.Brush, 91
operator double[]
VectSharp.Point, 630
operator false
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupExpre 547
operator Func< double, Colour >
VectSharp.Brush, 91

VectSharp.GradientStops, 295
operator OptionalContentGroupTreeLabel
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeLabel, 557
operator OptionalContentGroupTreeNode
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeNode, 561, 562
operator Point
VectSharp.Point, 630
operator SolidColourBrush
VectSharp.SolidColourBrush, 739
operator true
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupExpression, 547
operator!
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupExpression, 548
operator!=
VectSharp.Colour, 112
VectSharp.TextSpacing, 777
operator∗
VectSharp.Filters.ColourMatrix, 126
VectSharp.Point, 631
operator+
VectSharp.Point, 631
operator-
VectSharp.Point, 632
operator==
VectSharp.Colour, 112
VectSharp.TextSpacing, 777
operator&
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupExpression, 548
operator |
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupExpression, 548
OptionalContentGroup
VectSharp.PDF.OptionalContentGroups.OptionalContentGroup, 541
OptionalContentGroupAndExpression
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupAndExpression, 545
OptionalContentGroupBooleanOperation
VectSharp.PDF.OptionalContentGroups, 31
OptionalContentGroupNotExpression
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupNotExpression, 551
OptionalContentGroupOrExpression
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupOrExpression, 552
OptionalContentGroupSettings
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupSettings, 554
OptionalContentGroupTree
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupSettings, 554
OptionalContentGroupTreeLabel
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeLabel, 556
OptionalContentGroupTreeNode
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeNode, 559
Orange
VectSharp.Colours, 163
OrangeRed
VectSharp.Colours, 163
Orchid
VectSharp.Colours, 163
Origin
VectSharp.ThreeD.MaskedLightSource, 522
OuterRadius
VectSharp.Plots.Pie, 613
OutlineTree
VectSharp.PDF.OutlineTree, 564
OutlineTreeNode
VectSharp.PDF.OutlineTreeNode, 569
OutputFormats
VectSharp.Raster.ImageSharp, 34
OverallEasing
VectSharp.Transition, 781

Page
VectSharp.Markdown.PageFinishedEventArgs, 577
VectSharp.Markdown.PageStartedEventArgs, 579
VectSharp.Page, 573
PageFinished
VectSharp.Markdown.MarkdownRenderer, 516
PageFinishedEventArgs
VectSharp.Markdown.PageFinishedEventArgs, 577
PageHeight
VectSharp.Canvas.SKMultiLayerRenderCanvas, 714
PageMaxX
VectSharp.Markdown.LineEventArgs, 442
VectSharp.Markdown.LineMeasuredEventArgs, 449
PageMaxY
VectSharp.Markdown.LineEventArgs, 442
VectSharp.Markdown.LineMeasuredEventArgs, 449
Pages
VectSharp.Document, 220
VectSharp.PDF.PDFDocument, 596
PageSize
VectSharp.Markdown.MarkdownRenderer, 509
PageStarted
VectSharp.Markdown.MarkdownRenderer, 516
PageStartedEventArgs
VectSharp.Markdown.PageStartedEventArgs, 579
PageWidth
VectSharp.Canvas.SKMultiLayerRenderCanvas, 715
Paint
VectSharp.Canvas.SKRenderAction, 724
PaintToAnimatedCanvas

VectSharp.Canvas.AvaloniaContextInterpreter, 67
PaintToCanvas
VectSharp.Canvas.AvaloniaContextInterpreter, 67–69
PaintToSKCanvas
VectSharp.Canvas.SKRenderContextInterpreter, 731–734
PaleGoldenRod
VectSharp.Colours, 164
PaleGreen
VectSharp.Colours, 164
PaleTurquoise
VectSharp.Colours, 164
PaleVioletRed
VectSharp.Colours, 164
PapayaWhip
VectSharp.Colours, 164
ParallelLightSource
VectSharp.ThreeD.ParallelLightSource, 582
Parent
VectSharp.Canvas.RenderAction, 684
VectSharp.Canvas.SKRenderAction, 724
ParseImageURI
VectSharp.SVG.Parser, 587
Parser
VectSharp.ImageSharpUtils.ImageURIParser, 399
VectSharp.MuPDFUtils.ImageURIParser, 400
ParseSVGURI
VectSharp.SVG.Parser, 586
Pastel
VectSharp.Filters.ColourMatrix, 128
Path
VectSharp.Canvas.SKRenderAction, 724
VectSharp.Plots.PathDataPointElement, 590
path
VectSharp.Markdown.HTTPUtils, 361
PathAction
VectSharp.Canvas.RenderAction, 681
VectSharp.Canvas.SKRenderAction, 720
PathDataPointElement
VectSharp.Plots.PathDataPointElement, 589
Payload
VectSharp.Canvas.SKRenderAction, 725
PDFDocument
VectSharp.PDF.PDFDocument, 594
PDFMetadata
VectSharp.PDF.PDFMetadata, 597
PDFVersion
VectSharp.PDF.PDFDocument, 596
PeachPuff
VectSharp.Colours, 165
PenumbraAttenuationExponent
VectSharp.ThreeD.AreaLightSource, 65
PenumbraRadius
VectSharp.ThreeD.AreaLightSource, 65
Period
VectSharp.Plots.MovingAverageTrendLine, 528
Peru
VectSharp.Colours, 165
Phase
VectSharp.LineDash, 438
PhongMaterial
VectSharp.ThreeD.PhongMaterial, 607
Pie
VectSharp.Plots.Pie, 610, 611
Pink
VectSharp.Colours, 165
PixelFormats
VectSharp, 26
Plasma
VectSharp.Gradients, 289
PlasmaColouring
VectSharp.Gradients, 285
Plot
VectSharp.Plots.ActionDataPointElement, 36
VectSharp.Plots.Area< T >, 59
VectSharp.Plots.Bars< T >, 74
VectSharp.Plots.BoxPlot, 86
VectSharp.Plots.ClusteredBars, 102
VectSharp.Plots.ContinuousAxis, 181
VectSharp.Plots.ContinuousAxisLabels, 185
VectSharp.Plots.ContinuousAxisTicks, 189
VectSharp.Plots.ContinuousAxisTitle, 194
VectSharp.Plots.DataLabels< T >, 207
VectSharp.Plots.DataLine< T >, 212
VectSharp.Plots.ExponentialTrendLine, 223
VectSharp.Plots.Function2D, 270
VectSharp.Plots.GraphicsDataPointElement, 336
VectSharp.Plots.Grid, 358
VectSharp.Plots.IDataPointElement, 369
VectSharp.Plots.IPlotElement, 408
VectSharp.Plots.LinearTrendLine, 433
VectSharp.Plots.LogarithmicTrendLine, 469
VectSharp.Plots.MovingAverageTrendLine, 527
VectSharp.Plots.PathDataPointElement, 589
VectSharp.Plots.Pie, 611
VectSharp.Plots.Plot, 615
VectSharp.Plots.PlotElement< T >, 620
VectSharp.Plots.PolynomialTrendLine, 640
VectSharp.Plots.PowerLawTrendLine, 644
VectSharp.Plots.ScatterPoints< T >, 689
VectSharp.Plots.StackedBars, 750
VectSharp.Plots.Swarm, 762
VectSharp.Plots.TextLabel< T >, 772
VectSharp.Plots.Violin, 807
PlotAction
VectSharp.Plots.ActionDataPointElement, 37
VectSharp.Plots.PlotElement< T >, 621
PlotElement
VectSharp.Plots.PlotElement< T >, 620
PlotElementPresentationAttributes
VectSharp.Plots.PlotElementPresentationAttributes, 622
PlotElements
VectSharp.Plots.Plot, 618
PlotType

VectSharp.Plots.Function2D, 269
Plum
VectSharp.Colours, 165
PNGStream
VectSharp.RasterImage, 660
Point
VectSharp.Point, 625
VectSharp.Segment, 698
PointerEntered
VectSharp.Canvas.RenderAction, 685
VectSharp.Canvas.SKRenderAction, 726
PointerExited
VectSharp.Canvas.RenderAction, 685
VectSharp.Canvas.SKRenderAction, 726
PointerPressed
VectSharp.Canvas.RenderAction, 685
VectSharp.Canvas.SKRenderAction, 727
PointerReleased
VectSharp.Canvas.RenderAction, 685
VectSharp.Canvas.SKRenderAction, 727
PointLightSource
VectSharp.ThreeD.PointLightSource, 634
PointMargin
VectSharp.Plots.Swarm, 763
Points
VectSharp.Segment, 698
PointSize
VectSharp.Plots.Swarm, 764
PolynomialTrendLine
VectSharp.Plots.PolynomialTrendLine, 638, 639
Position
VectSharp.Font.FontUnderline, 258
VectSharp.Markdown.TextFragment, 770
VectSharp.Plots.BoxPlot, 88
VectSharp.Plots.ContinuousAxisLabels, 186
VectSharp.Plots.ContinuousAxisTitle, 196
VectSharp.Plots.Swarm, 764
VectSharp.Plots.TextLabel< T >, 773
VectSharp.Plots.Violin, 808
VectSharp.ThreeD.MaskedLightSource, 522
VectSharp.ThreeD.PointLightSource, 636
VectSharp.ThreeD.SpotlightLightSource, 747
PowderBlue
VectSharp.Colours, 165
PowerLawTrendLine
VectSharp.Plots.PowerLawTrendLine, 643, 644
PresentationAttributes
VectSharp.Plots.Area< T >, 60
VectSharp.Plots.Bars< T >, 75
VectSharp.Plots.ClusteredBars, 103
VectSharp.Plots.ContinuousAxis, 182
VectSharp.Plots.ContinuousAxisLabels, 186
VectSharp.Plots.ContinuousAxisTicks, 190
VectSharp.Plots.ContinuousAxisTitle, 196
VectSharp.Plots.DataLabels< T >, 209
VectSharp.Plots.DataLine< T >, 212
VectSharp.Plots.ExponentialTrendLine, 224
VectSharp.Plots.Grid, 359
VectSharp.Plots.LinearTrendLine, 435
VectSharp.Plots.LogarithmicTrendLine, 471
VectSharp.Plots.MovingAverageTrendLine, 528
VectSharp.Plots.Pie, 613
VectSharp.Plots.PolynomialTrendLine, 641
VectSharp.Plots.PowerLawTrendLine, 646
VectSharp.Plots.ScatterPoints< T >, 690
VectSharp.Plots.StackedBars, 751
VectSharp.Plots.Swarm, 764
VectSharp.Plots.TextLabel< T >, 774
VectSharp.Plots.Violin, 808
PreserveAlpha
VectSharp.Filters.ConvolutionFilter, 200
Producer
VectSharp.PDF.PDFMetadata, 600
Purple
VectSharp.Colours, 166

QuadraticBezierTo
VectSharp.GraphicsPath, 354
QuoteBlockBackgroundColour
VectSharp.Markdown.MarkdownRenderer, 509
QuoteBlockBarColour
VectSharp.Markdown.MarkdownRenderer, 509
QuoteBlockBarWidth
VectSharp.Markdown.MarkdownRenderer, 509
QuoteBlockIndentWidth
VectSharp.Markdown.MarkdownRenderer, 510

R
VectSharp.Colour, 116
VectSharp.SolidColourBrush, 740
R1
VectSharp.Filters.ColourMatrix, 131
R2
VectSharp.Filters.ColourMatrix, 132
R3
VectSharp.Filters.ColourMatrix, 132
R4
VectSharp.Filters.ColourMatrix, 132
R5
VectSharp.Filters.ColourMatrix, 132
RadialGradientBrush
VectSharp.RadialGradientBrush, 648
RadioButtonGroups
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupSettin
554
Radius
VectSharp.RadialGradientBrush, 650
VectSharp.ThreeD.AreaLightSource, 65
Rainbow
VectSharp.Gradients, 289
RangeRecord
VectSharp.TrueTypeFile.CoverageTable.RangeRecord,
651
RasterImage
VectSharp.RasterImage, 657, 658
RasterImageFile
VectSharp.ImageSharpUtils.RasterImageFile, 662

VectSharp.MuPDFUtils.RasterImageFile, 663
RasterImageLoader
VectSharp.Markdown.MarkdownRenderer, 510
RasterImageStream
VectSharp.ImageSharpUtils.RasterImageStream, 664, 665
VectSharp.MuPDFUtils.RasterImageStream, 666, 667
RasterisationMethod
VectSharp.Graphics, 334
RasterisationResolution
VectSharp.Canvas.FilterOption, 227
VectSharp.PDF.PDFContextInterpreter.FilterOption, 230
VectSharp.SVG.SVGContextInterpreter.FilterOption, 233
RasterisationResolutionRelative
VectSharp.Canvas.FilterOption, 227
VectSharp.PDF.PDFContextInterpreter.FilterOption, 230
VectSharp.SVG.SVGContextInterpreter.FilterOption, 233
Rasterise
VectSharp.Canvas.SKRenderContextInterpreter, 736
VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter, 392
VectSharp.Raster.Raster, 652
RasteriseParameter
VectSharp.Filters.FilterWithRasterisableParameter, 235
VectSharp.Filters.IFilterWithRasterisableParameter, 375
RasterResolutionX
VectSharp.Plots.Function2D, 271
RasterResolutionY
VectSharp.Plots.Function2D, 271
RebeccaPurple
VectSharp.Colours, 166
Rectangle
VectSharp.IGraphicsContext, 382
VectSharp.Rectangle, 669, 670
RectangleFragment
VectSharp.Markdown.RectangleFragment, 675
Red
VectSharp.Colours, 166
RedToGreen
VectSharp.Gradients, 289
RedYellowGreen
VectSharp.Gradients, 290
RegularFontFamily
VectSharp.Markdown.MarkdownRenderer, 510
RelativeTo
VectSharp.LinearGradientBrush, 430
Remove
VectSharp.PDF.AnnotationStyleCollection, 54
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeNode, 562
VectSharp.PDF.OutlineTree, 566
VectSharp.PDF.OutlineTreeNode, 570
RemoveAt
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeN 562
VectSharp.PDF.OutlineTree, 566
VectSharp.PDF.OutlineTreeNode, 571
RemoveLastFrame
VectSharp.Animation, 48
RemoveLayer
VectSharp.Canvas.SKMultiLayerRenderCanvas, 711
RemovePlotElement
VectSharp.Plots.Plot, 617
Render
VectSharp.Canvas.AnimatedCanvas, 41
VectSharp.Canvas.SKMultiLayerRenderCanvas, 712
VectSharp.Markdown.LineFragment, 444
VectSharp.Markdown.MarkdownRenderer, 499
VectSharp.Markdown.RectangleFragment, 676
VectSharp.Markdown.TextFragment, 769
VectSharp.Markdown.UnderlineFragment, 801
VectSharp.Plots.Plot, 617, 618
RenderActions
VectSharp.Canvas.SKMultiLayerRenderCanvas, 714
RenderAtResolution
VectSharp.Canvas.SKMultiLayerRenderCanvas, 712
Renderer
VectSharp.MarkdownCanvas.MarkdownCanvasControl, 485
RenderLock
VectSharp.Canvas.SKMultiLayerRenderCanvas, 714
RenderSinglePage
VectSharp.Markdown.MarkdownRenderer, 500
RepeatCount
VectSharp.Animation, 49
Replace
VectSharp.ThreeD.IScene, 410, 411
VectSharp.ThreeD.Scene, 693
Resolution
VectSharp.Plots.IContinuousCoordinateSystem, 364
VectSharp.Plots.LinearCoordinateSystem2D, 425
VectSharp.Plots.LinLogCoordinateSystem2D, 456
VectSharp.Plots.LogarithmicCoordinateSystem2D, 466
VectSharp.Plots.LogLinCoordinateSystem2D, 478
ResolveEmojiUsingOpenMoji
VectSharp.Markdown.HTTPUtils, 361
ResolveFontFamily
VectSharp.DefaultFontLibrary, 214
VectSharp.FolderFontLibrary, 237, 238
VectSharp.MemoryFontLibrary, 248, 249
VectSharp.FontLibrary, 255, 256

VectSharp.IFontLibrary, 376, 377
VectSharp.MultiFontLibrary, 531
VectSharp.SimpleFontLibrary, 704
ResolveImageURI
VectSharp.Markdown.HTTPUtils, 361
ResourceFontFamily
VectSharp.ResourceFontFamily, 687
ResourceName
VectSharp.ResourceFontFamily, 687
Restore
VectSharp.Graphics, 320
RestoreAction
VectSharp.Canvas.SKRenderAction, 721
Reverse
VectSharp.GraphicsPath, 355
ReverseDirection
VectSharp.ThreeD.ParallelLightSource, 584
Right
VectSharp.Markdown.Margins, 480
RightSideBearing
VectSharp.Font.DetailedFontMetrics, 216
VectSharp.TrueTypeFile.Bearings, 77
Rocket
VectSharp.Gradients, 290
RocketColouring
VectSharp.Gradients, 286
RosyBrown
VectSharp.Colours, 166
Rotate
VectSharp.Graphics, 320
VectSharp.IGraphicsContext, 383
RotateAt
VectSharp.Graphics, 320
Rotation
VectSharp.Plots.ContinuousAxisLabels, 186
VectSharp.Plots.ContinuousAxisTitle, 196
VectSharp.Plots.DataLabels< T >, 209
VectSharp.Plots.TextLabel< T >, 774
RoyalBlue
VectSharp.Colours, 166

S
VectSharp.Colour, 117
SaddleBrown
VectSharp.Colours, 167
Salmon
VectSharp.Colours, 167
SampledPointElement
VectSharp.Plots.Function2D, 272
SandyBrown
VectSharp.Colours, 167
Save
VectSharp.Graphics, 321
SaveAction
VectSharp.Canvas.SKRenderAction, 721
SaveAsAnimatedGIF
VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter, 393, 394
SaveAsAnimatedPNG
VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter, 395
VectSharp.Raster.Raster, 653
SaveAsAnimatedSVG
VectSharp.SVG.SVGContextInterpreter, 754, 755
SaveAsAnimatedSVGWithFrames
VectSharp.SVG.SVGContextInterpreter, 756, 757
SaveAsImage
VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter, 396, 397
SaveAsJPEG
VectSharp.Raster.Raster, 654
SaveAsPDF
VectSharp.PDF.PDFContextInterpreter, 592
SaveAsPNG
VectSharp.Raster.Raster, 654, 655
SaveAsRawBytes
VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter, 397, 398
VectSharp.Raster.Raster, 655
SaveAsSVG
VectSharp.SVG.SVGContextInterpreter, 758, 759
Scale
VectSharp.Filters.ConvolutionFilter, 200
VectSharp.Graphics, 321
VectSharp.IGraphicsContext, 383
VectSharp.Plots.LinearCoordinateSystem1D, 417
VectSharp.Plots.LogarithmicCoordinateSystem1D, 459
ScaleX
VectSharp.Plots.LinearCoordinateSystem2D, 425
VectSharp.Plots.LinLogCoordinateSystem2D, 456
VectSharp.Plots.LogarithmicCoordinateSystem2D, 466
VectSharp.Plots.LogLinCoordinateSystem2D, 478
ScaleY
VectSharp.Plots.LinearCoordinateSystem2D, 425
VectSharp.Plots.LinLogCoordinateSystem2D, 456
VectSharp.Plots.LogarithmicCoordinateSystem2D, 466
VectSharp.Plots.LogLinCoordinateSystem2D, 478
ScatterPoints
VectSharp.Plots.ScatterPoints< T >, 688
Scene
VectSharp.ThreeD.Scene, 692
SceneElements
VectSharp.ThreeD.IScene, 411
VectSharp.ThreeD.Scene, 694
SceneLock
VectSharp.ThreeD.IScene, 411
VectSharp.ThreeD.Scene, 694
Script
VectSharp, 26
VectSharp.FormattedText, 265
SeaGreen
VectSharp.Colours, 167
SeaShell
VectSharp.Colours, 167

Segments
VectSharp.GraphicsPath, 356
SegmentType
VectSharp, 26
SendToBack
VectSharp.Canvas.RenderAction, 681
SetClippingPath
VectSharp.Graphics, 321, 322
SetFillStyle
VectSharp.IGraphicsContext, 383, 384
SetLineDash
VectSharp.IGraphicsContext, 384
SetStrokeStyle
VectSharp.IGraphicsContext, 384
ShadowSamplingPointCount
VectSharp.ThreeD.AreaLightSource, 65
Side1End
VectSharp.Plots.Grid, 359
Side1Start
VectSharp.Plots.Grid, 359
Side2End
VectSharp.Plots.Grid, 360
Side2Start
VectSharp.Plots.Grid, 360
Sides
VectSharp.Plots.Violin, 809
Sienna
VectSharp.Colours, 168
Silver
VectSharp.Colours, 168
SimpleFontLibrary
VectSharp.SimpleFontLibrary, 700–702
Size
VectSharp.Markdown.RectangleFragment, 677
VectSharp.Plots.ScatterPoints< T >, 690
VectSharp.Rectangle, 673
VectSharp.Size, 705
SizeAbove
VectSharp.Plots.ContinuousAxisTicks, 191
SizeBelow
VectSharp.Plots.ContinuousAxisTicks, 191
SkipDescenders
VectSharp.Font.FontUnderline, 258
SKMultiLayerRenderCanvas
VectSharp.Canvas.SKMultiLayerRenderCanvas, 708, 709
SkyBlue
VectSharp.Colours, 168
SlateBlue
VectSharp.Colours, 168
SlateGray
VectSharp.Colours, 168
SlateGrey
VectSharp.Colours, 169
Slope
VectSharp.Plots.ExponentialTrendLine, 224
VectSharp.Plots.LinearTrendLine, 435
VectSharp.Plots.LogarithmicTrendLine, 471
VectSharp.Plots.PowerLawTrendLine, 646
Smooth
VectSharp.Plots.Area< T >, 60
VectSharp.Plots.DataLine< T >, 213
VectSharp.Plots.Violin, 809
Snow
VectSharp.Colours, 169
SolidColourBrush
VectSharp.SolidColourBrush, 738
SolidLine
VectSharp.LineDash, 438
SourceDistance
VectSharp.ThreeD.AreaLightSource, 65
SpaceAfterHeading
VectSharp.Markdown.MarkdownRenderer, 510
SpaceAfterLine
VectSharp.Markdown.MarkdownRenderer, 510
SpaceAfterParagraph
VectSharp.Markdown.MarkdownRenderer, 511
SpaceBeforeHeading
VectSharp.Markdown.MarkdownRenderer, 511
SpaceBeforeParagaph
VectSharp.Markdown.MarkdownRenderer, 511
Spacing
VectSharp.FormattedText, 265
SpecularReflectionCoefficient
VectSharp.ThreeD.PhongMaterial, 609
SpecularShininess
VectSharp.ThreeD.PhongMaterial, 609
SplineEasing
VectSharp.SplineEasing, 741
SpotlightLightSource
VectSharp.ThreeD.SpotlightLightSource, 744
SpringGreen
VectSharp.Colours, 169
StackedBars
VectSharp.Plots.StackedBars, 748, 749
StandardDeviation
VectSharp.Filters.GaussianBlurFilter, 280
StandardFamilies
VectSharp.FontFamily, 250
StandardFontFamilies
VectSharp.FontFamily, 246
StandardFontFamilyResources
VectSharp.FontFamily, 250
Start
VectSharp.Markdown.UnderlineFragment, 803
StartAngle
VectSharp.Plots.Pie, 613
StartCoverageIndex
VectSharp.TrueTypeFile.CoverageTable.RangeRecord, 651
StartGlyphID
VectSharp.TrueTypeFile.ClassDefinitionTable.ClassRangeRecord, 98
VectSharp.TrueTypeFile.CoverageTable.RangeRecord, 651
StartNewPage

VectSharp.Markdown.MarkdownContext, 489
StartPoint
VectSharp.LinearGradientBrush, 430
VectSharp.Plots.ContinuousAxis, 182
VectSharp.Plots.ContinuousAxisLabels, 187
VectSharp.Plots.ContinuousAxisTicks, 191
VectSharp.Plots.ContinuousAxisTitle, 196
SteelBlue
VectSharp.Colours, 169
StepsX
VectSharp.Plots.Function2DGrid, 277
StepsY
VectSharp.Plots.Function2DGrid, 277
StopTolerance
VectSharp.GradientStops, 296
StrikeThrough
VectSharp.Markdown.MarkdownContext, 492
Stroke
VectSharp.Canvas.RenderAction, 684
VectSharp.Plots.PlotElementPresentationAttributes, 623
StrokePath
VectSharp.Graphics, 322
StrokeRectangle
VectSharp.Graphics, 323, 324
StrokeStyle
VectSharp.IGraphicsContext, 387
StrokeText
VectSharp.Graphics, 324–327
VectSharp.IGraphicsContext, 385
StrokeTextOnPath
VectSharp.Graphics, 328
StrokeTextUnderline
VectSharp.Graphics, 329–331
Styles
VectSharp.PDF.AnnotationStyleCollection, 56
Subject
VectSharp.PDF.PDFMetadata, 600
SubscriptShift
VectSharp.Markdown.MarkdownRenderer, 511
SubsetFont
VectSharp.TrueTypeFile, 794
SubSuperscriptFontSize
VectSharp.Markdown.MarkdownRenderer, 511
SuperscriptShift
VectSharp.Markdown.MarkdownRenderer, 512
Swarm
VectSharp.Plots.Swarm, 762
SwarmPointElement
VectSharp.Plots.Swarm, 764
SwitchLayers
VectSharp.Canvas.SKMultiLayerRenderCanvas, 712
SyntaxHighlighter
VectSharp.Markdown.MarkdownRenderer, 512

TableCellMargins
VectSharp.Markdown.MarkdownRenderer, 512
TableHeaderRowSeparatorColour
VectSharp.Markdown.MarkdownRenderer, 512
TableHeaderRowSeparatorThickness
VectSharp.Markdown.MarkdownRenderer, 513
TableHeaderSeparatorThickness
VectSharp.Markdown.MarkdownRenderer, 513
TableRowSeparatorColour
VectSharp.Markdown.MarkdownRenderer, 513
TableVAlign
VectSharp.Markdown.MarkdownRenderer, 513
Tag
VectSharp.Canvas.RenderAction, 684
VectSharp.Canvas.SKRenderAction, 725
VectSharp.IGraphicsContext, 387
VectSharp.Markdown.LineFragment, 445
VectSharp.Markdown.MarkdownContext, 492
VectSharp.Plots.Area< T >, 60
VectSharp.Plots.Bars< T >, 76
VectSharp.Plots.BoxPlot, 89
VectSharp.Plots.ClusteredBars, 103
VectSharp.Plots.ContinuousAxis, 183
VectSharp.Plots.ContinuousAxisLabels, 187
VectSharp.Plots.ContinuousAxisTicks, 191
VectSharp.Plots.ContinuousAxisTitle, 196
VectSharp.Plots.DataLabels< T >, 210
VectSharp.Plots.DataLine< T >, 213
VectSharp.Plots.ExponentialTrendLine, 225
VectSharp.Plots.Function2D, 272
VectSharp.Plots.Grid, 360
VectSharp.Plots.LinearTrendLine, 435
VectSharp.Plots.LogarithmicTrendLine, 471
VectSharp.Plots.MovingAverageTrendLine, 528
VectSharp.Plots.Pie, 613
VectSharp.Plots.PolynomialTrendLine, 641
VectSharp.Plots.PowerLawTrendLine, 646
VectSharp.Plots.ScatterPoints< T >, 690
VectSharp.Plots.StackedBars, 751
VectSharp.Plots.Swarm, 764
VectSharp.Plots.TextLabel< T >, 774
VectSharp.Plots.Violin, 809
Tan
VectSharp.Colours, 169
TaskListCheckedBullet
VectSharp.Markdown.MarkdownRenderer, 513
TaskListUncheckedBullet
VectSharp.Markdown.MarkdownRenderer, 514
Teal
VectSharp.Colours, 170
Text
VectSharp.Canvas.RenderAction, 684
VectSharp.Canvas.SKRenderAction, 725
VectSharp.FormattedText, 265
VectSharp.Markdown.FormattedString, 260
VectSharp.Markdown.TextFragment, 770
TextAction
VectSharp.Canvas.RenderAction, 681
VectSharp.Canvas.SKRenderAction, 721
TextAnchors
VectSharp, 26

TextBaseline
VectSharp.IGraphicsContext, 387
TextBaselines
VectSharp, 27
TextConversionOption
VectSharp.MarkdownCanvas.MarkdownCanvasControl, 485
TextConversionOptionsProperty
VectSharp.MarkdownCanvas.MarkdownCanvasControl, 484
TextFormat
VectSharp.Plots.ContinuousAxisLabels, 187
TextFragment
VectSharp.Markdown.TextFragment, 767
TextLabel
VectSharp.Plots.TextLabel< T >, 772
TextOptions
VectSharp.Canvas.AvaloniaContextInterpreter, 67
VectSharp.PDF.PDFContextInterpreter, 591
VectSharp.SVG.SVGContextInterpreter, 753
TextSpacing
VectSharp.TextSpacing, 776
TextX
VectSharp.Canvas.SKRenderAction, 725
TextY
VectSharp.Canvas.SKRenderAction, 725
ThematicBreakLineColour
VectSharp.Markdown.MarkdownRenderer, 514
ThematicBreakThickness
VectSharp.Markdown.MarkdownRenderer, 514
Thickness
VectSharp.Font.FontUnderline, 258
VectSharp.Markdown.UnderlineFragment, 803
this[int index]
VectSharp.GradientStops, 296
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupCategory, 563
VectSharp.PDF.OutlineTree, 567
VectSharp.PDF.OutlineTreeNode, 572
VectSharp.Point, 633
this[int y, int x]
VectSharp.Filters.ColourMatrix, 132
this[string key]
VectSharp.PDF.AnnotationStyleCollection, 56
Thistle
VectSharp.Colours, 170
Title
VectSharp.PDF.OutlineTreeNode, 572
VectSharp.PDF.PDFMetadata, 600
VectSharp.Plots.ContinuousAxisTitle, 197
ToColour
VectSharp.Filters.ColourMatrix, 126
ToCSSString
VectSharp.Colour, 113
ToDataCoordinates
VectSharp.Plots.IContinuousInvertibleCoordinateSystem, 365
VectSharp.Plots.LinearCoordinateSystem2D, 422
VectSharp.Plots.LinLogCoordinateSystem2D, 454
VectSharp.Plots.LogarithmicCoordinateSystem2D, 464
VectSharp.Plots.LogLinCoordinateSystem2D, 476
ToHSL
VectSharp.Colour, 113
ToLab
VectSharp.Colour, 113
Tomato
VectSharp.Colours, 170
Top
VectSharp.Font.DetailedFontMetrics, 217
VectSharp.Markdown.Margins, 480
TopLeft
VectSharp.Markdown.RectangleFragment, 678
TopLeftMargin
VectSharp.Filters.BoxBlurFilter, 83
VectSharp.Filters.ColourMatrixFilter, 135
VectSharp.Filters.CompositeLocationInvariantFilter, 177
VectSharp.Filters.ConvolutionFilter, 201
VectSharp.Filters.GaussianBlurFilter, 280
VectSharp.Filters.IFilter, 372
VectSharp.Filters.MaskFilter, 524
TopLevelItems
VectSharp.PDF.OutlineTree, 567
ToPlotCoordinates
VectSharp.Plots.CategoricalCoordinateSystem1D< T >, 96
VectSharp.Plots.CompositeCoordinateSystem2D< T1, T2 >, 173
VectSharp.Plots.CoordinateSystem< T >, 202
VectSharp.Plots.CoordinateSystem1D< T >, 205
VectSharp.Plots.ICoordinateSystem< T >, 367
VectSharp.Plots.ICoordinateSystem1D< T >, 368
VectSharp.Plots.LinearCoordinateSystem1D, 417
VectSharp.Plots.LinearCoordinateSystem2D, 422
VectSharp.Plots.LinLogCoordinateSystem2D, 454
VectSharp.Plots.LogarithmicCoordinateSystem1D, 458
VectSharp.Plots.LogarithmicCoordinateSystem2D, 464
VectSharp.Plots.LogLinCoordinateSystem2D, 476
ToRectangular
VectSharp.Plots.Function2DGrid, 275
ToString
VectSharp.PDF.OptionalContentGroups.OptionalContentGroup, 542
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupExpre
549
ToXYZ
VectSharp.Colour, 113
Transform
VectSharp.Canvas.RenderAction, 685
VectSharp.Canvas.SKRenderAction, 726
VectSharp.Graphics, 331, 332
VectSharp.GraphicsPath, 355
VectSharp.IGraphicsContext, 385

VectSharp.Segment, 698
TransformAction
VectSharp.Canvas.SKRenderAction, 722
Transition
VectSharp.Transition, 780
Transitions
VectSharp.Animation, 49
Translate
VectSharp.Graphics, 333
VectSharp.IGraphicsContext, 386
VectSharp.Markdown.LineFragment, 445
VectSharp.Markdown.RectangleFragment, 677
VectSharp.Markdown.TextFragment, 769
VectSharp.Markdown.UnderlineFragment, 802
Translation
VectSharp.Markdown.MarkdownContext, 493
TransparentToBlack
VectSharp.Gradients, 290
Triangulate
VectSharp.GraphicsPath, 356
TrueTypeFile
VectSharp.FontFamily, 252
TryGetValue
VectSharp.PDF.AnnotationStyleCollection, 55
TryRasterise
VectSharp.Graphics, 334
Turbo
VectSharp.Gradients, 290
TurboColouring
VectSharp.Gradients, 286
Turquoise
VectSharp.Colours, 170
Type
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupFreeLabel, 556
VectSharp.Plots.Function2D, 272
VectSharp.Plots.Function2DGrid, 277
VectSharp.Segment, 699

UnbalancedStackAction
VectSharp.Graphics, 335
UnbalancedStackActions
VectSharp, 27
Underline
VectSharp.Font, 244
VectSharp.Markdown.MarkdownContext, 493
UnderlineFragment
VectSharp.Markdown.UnderlineFragment, 800
UnderlineThickness
VectSharp.Markdown.MarkdownRenderer, 515
Union
VectSharp.Rectangle, 671, 672
UnitsOff
VectSharp.LineDash, 438
UnitsOn
VectSharp.LineDash, 438
UpdateLayer
VectSharp.Canvas.SKMultiLayerRenderCanvas, 713
UpdateWith
VectSharp.Canvas.SKMultiLayerRenderCanvas, 713
UseUnicodeEmoji
VectSharp.Markdown.MarkdownEmojiExtension, 493
UseUniqueTags
VectSharp.Graphics, 335

Values
VectSharp.PDF.AnnotationStyleCollection, 56
VectSharp, 23
FillRule, 25
LineCaps, 25
LineJoins, 26
PixelFormats, 26
Script, 26
SegmentType, 26
TextAnchors, 26
TextBaselines, 27
UnbalancedStackActions, 27
VectSharp.AnimatedPNG, 44
Create, 45
InterframeCompression, 44
VectSharp.AnimatedPNG.CompressedFrame, 177
CompressedFrame, 178
Dispose, 179
Duration, 179
VectSharp.Animation, 45
AddFrame, 47
Animation, 46
Background, 48
Duration, 48
Frames, 49
GetFrameAtAbsolute, 47
GetFrameAtRelative, 48
Height, 49
LinearisationResolution, 49
RemoveLastFrame, 48
RepeatCount, 49
Transitions, 49
Width, 50
VectSharp.Brush, 90
MultiplyOpacity, 91
operator Brush, 91
VectSharp.Canvas, 27
VectSharp.Canvas.AnimatedCanvas, 40
CurrentFrame, 43
CurrentFrameProperty, 41
FrameCount, 43
FrameCountProperty, 42
FrameRate, 43
FrameRateProperty, 42
IsPlaying, 43
IsPlayingProperty, 42
Render, 41
VectSharp.Canvas.AvaloniaContextInterpreter, 66
PaintToAnimatedCanvas, 67
PaintToCanvas, 67–69

TextOptions, 67
VectSharp.Canvas.FilterOption, 225
Default, 227
FilterOperations, 226
FilterOption, 226
Operation, 227
RasterisationResolution, 227
RasterisationResolutionRelative, 227
VectSharp.Canvas.RenderAction, 678
ActionType, 682
ActionTypes, 680
BringToFront, 680
ClippingPath, 682
Fill, 682
Geometry, 683
ImageAction, 680
ImageDestination, 683
ImageId, 683
ImageSource, 683
InverseTransform, 683
Layout, 684
Parent, 684
PathAction, 681
PointerEntered, 685
PointerExited, 685
PointerPressed, 685
PointerReleased, 685
SendToBack, 681
Stroke, 684
Tag, 684
Text, 684
TextAction, 681
Transform, 685
VectSharp.Canvas.SKMultiLayerRenderCanvas, 706
AddLayer, 710
ClipMargin, 714
Dispose, 710
InsertLayer, 710
InvalidateDirty, 710
InvalidateZIndex, 711
LayerTransforms, 714
MoveLayer, 711
PageHeight, 714
PageWidth, 715
RemoveLayer, 711
Render, 712
RenderActions, 714
RenderAtResolution, 712
RenderLock, 714
SKMultiLayerRenderCanvas, 708, 709
SwitchLayers, 712
UpdateLayer, 713
UpdateWith, 713
VectSharp.Canvas.SKRenderAction, 715
ActionType, 722
ActionTypes, 717
ClipAction, 718
Dispose, 718
Disposed, 723
DrawFilteredGraphicsAction, 718
Filter, 723
Font, 723
Graphics, 723
ImageAction, 718
ImageDestination, 723
ImageId, 724
ImageSource, 724
InvalidateAll, 719
InvalidateHitTestPath, 719
InvalidateVisual, 719
InvalidateZIndex, 720
Paint, 724
Parent, 724
Path, 724
PathAction, 720
Payload, 725
PointerEntered, 726
PointerExited, 726
PointerPressed, 727
PointerReleased, 727
RestoreAction, 721
SaveAction, 721
Tag, 725
Text, 725
TextAction, 721
TextX, 725
TextY, 725
Transform, 726
TransformAction, 722
ZIndex, 726
VectSharp.Canvas.SKRenderContext, 727
VectSharp.Canvas.SKRenderContextInterpreter, 728
CopyToSKRenderContext, 729, 730
PaintToSKCanvas, 731–734
Rasterise, 736
VectSharp.Canvas/AnimatedCanvas.cs, 811
VectSharp.Canvas/AvaloniaContext.cs, 815
VectSharp.Canvas/RenderingParameters.cs, 850
VectSharp.Canvas/SkiaBitmap.cs, 852
VectSharp.Canvas/SKMultiLayerRenderCanvas.cs, 853
VectSharp.Canvas/SKRenderContext.cs, 868
VectSharp.Colour, 104
A, 115
a, 115
B, 116
Equals, 106
FromCSSString, 106
FromHSL, 106
FromLab, 107
FromRgb, 107, 108
FromRgba, 109–111
FromXYZ, 112
G, 116
GetHashCode, 112
H, 116
L, 116

operator!=, 112
operator==, 112
R, 116
S, 117
ToCSSString, 113
ToHSL, 113
ToLab, 113
ToXYZ, 113
WithAlpha, 114, 115
X, 117
Y, 117
VectSharp.Colours, 136
AliceBlue, 142
AntiqueWhite, 142
Aqua, 142
Aquamarine, 143
Azure, 143
Beige, 143
Bisque, 143
Black, 143
BlanchedAlmond, 144
Blue, 144
BlueViolet, 144
Brown, 144
BurlyWood, 144
CadetBlue, 145
Chartreuse, 145
Chocolate, 145
Coral, 145
CornflowerBlue, 145
Cornsilk, 146
Crimson, 146
Cyan, 146
DarkBlue, 146
DarkCyan, 146
DarkGoldenRod, 147
DarkGray, 147
DarkGreen, 147
DarkGrey, 147
DarkKhaki, 147
DarkMagenta, 148
DarkOliveGreen, 148
DarkOrange, 148
DarkOrchid, 148
DarkRed, 148
DarkSalmon, 149
DarkSeaGreen, 149
DarkSlateBlue, 149
DarkSlateGray, 149
DarkSlateGrey, 149
DarkTurquoise, 150
DarkViolet, 150
DeepPink, 150
DeepSkyBlue, 150
DimGray, 150
DimGrey, 151
DodgerBlue, 151
FireBrick, 151
FloralWhite, 151
ForestGreen, 151
Fuchsia, 152
Gainsboro, 152
GhostWhite, 152
Gold, 152
GoldenRod, 152
Gray, 153
Green, 153
GreenYellow, 153
Grey, 153
HoneyDew, 153
HotPink, 154
IndianRed, 154
Indigo, 154
Ivory, 154
Khaki, 154
Lavender, 155
LavenderBlush, 155
LawnGreen, 155
LemonChiffon, 155
LightBlue, 155
LightCoral, 156
LightCyan, 156
LightGoldenRodYellow, 156
LightGray, 156
LightGreen, 156
LightGrey, 157
LightPink, 157
LightSalmon, 157
LightSeaGreen, 157
LightSkyBlue, 157
LightSlateGray, 158
LightSlateGrey, 158
LightSteelBlue, 158
LightYellow, 158
Lime, 158
LimeGreen, 159
Linen, 159
Magenta, 159
Maroon, 159
MediumAquaMarine, 159
MediumBlue, 160
MediumOrchid, 160
MediumPurple, 160
MediumSeaGreen, 160
MediumSlateBlue, 160
MediumSpringGreen, 161
MediumTurquoise, 161
MediumVioletRed, 161
MidnightBlue, 161
MintCream, 161
MistyRose, 162
Moccasin, 162
NavajoWhite, 162
Navy, 162
OldLace, 162
Olive, 163

OliveDrab, 163
Orange, 163
OrangeRed, 163
Orchid, 163
PaleGoldenRod, 164
PaleGreen, 164
PaleTurquoise, 164
PaleVioletRed, 164
PapayaWhip, 164
PeachPuff, 165
Peru, 165
Pink, 165
Plum, 165
PowderBlue, 165
Purple, 166
RebeccaPurple, 166
Red, 166
RosyBrown, 166
RoyalBlue, 166
SaddleBrown, 167
Salmon, 167
SandyBrown, 167
SeaGreen, 167
SeaShell, 167
Sienna, 168
Silver, 168
SkyBlue, 168
SlateBlue, 168
SlateGray, 168
SlateGrey, 169
Snow, 169
SpringGreen, 169
SteelBlue, 169
Tan, 169
Teal, 170
Thistle, 170
Tomato, 170
Turquoise, 170
Violet, 170
Wheat, 171
White, 171
WhiteSmoke, 171
Yellow, 171
YellowGreen, 171
VectSharp.DefaultFontLibrary, 213
ResolveFontFamily, 214
VectSharp.DisposableIntPtr, 217
DisposableIntPtr, 218
Dispose, 218
InternalPointer, 219
VectSharp.Document, 219
Document, 219
Pages, 220
VectSharp.Filters, 28
VectSharp.Filters.BoxBlurFilter, 81
BottomRightMargin, 83
BoxBlurFilter, 82
BoxRadius, 83
Filter, 82
TopLeftMargin, 83
VectSharp.Filters.ColourMatrix, 120
A1, 128
A2, 129
A3, 129
A4, 129
A5, 129
AlphaInversion, 127
Apply, 122–124
B1, 129
B2, 130
B3, 130
B4, 130
B5, 130
ColourMatrix, 122
G1, 130
G2, 131
G3, 131
G4, 131
G5, 131
GreyScale, 127
Identity, 127
Inversion, 128
InvertedAlphaShift, 128
LuminanceToAlpha, 125
LuminanceToColour, 125
operator∗, 126
Pastel, 128
R1, 131
R2, 132
R3, 132
R4, 132
R5, 132
this[int y, int x], 132
ToColour, 126
WithAlpha, 127
VectSharp.Filters.ColourMatrixFilter, 133
BottomRightMargin, 135
ColourMatrix, 135
ColourMatrixFilter, 134
Filter, 134
TopLeftMargin, 135
VectSharp.Filters.CompositeLocationInvariantFilter, 174
BottomRightMargin, 176
CompositeLocationInvariantFilter, 175
Filter, 176
Filters, 176
TopLeftMargin, 177
VectSharp.Filters.ConvolutionFilter, 197
Bias, 199
BottomRightMargin, 200
ConvolutionFilter, 198
Filter, 199
Kernel, 200
Normalisation, 200
PreserveAlpha, 200
Scale, 200

TopLeftMargin, 201
VectSharp.Filters.FilterWithRasterisableParameter, 234
Dispose, 234
RasteriseParameter, 235
VectSharp.Filters.GaussianBlurFilter, 278
BottomRightMargin, 280
Filter, 279
GaussianBlurFilter, 279
StandardDeviation, 280
TopLeftMargin, 280
VectSharp.Filters.IFilter, 371
BottomRightMargin, 372
TopLeftMargin, 372
VectSharp.Filters.IFilterWithLocation, 373
Filter, 373
VectSharp.Filters.IFilterWithRasterisableParameter, 374
RasteriseParameter, 375
VectSharp.Filters.ILocationInvariantFilter, 390
Filter, 391
VectSharp.Filters.MaskFilter, 522
BottomRightMargin, 524
Filter, 523
Mask, 524
MaskFilter, 523
TopLeftMargin, 524
VectSharp.FolderFontLibrary, 235
FolderFontLibrary, 237
ResolveFontFamily, 237, 238
VectSharp.Font, 238
Ascent, 243
Descent, 243
EnableKerning, 243
Font, 240
FontFamily, 243
FontSize, 243
MeasureText, 242
MeasureTextAdvanced, 242
Underline, 244
WinAscent, 244
YMax, 244
YMin, 244
VectSharp.Font.DetailedFontMetrics, 215
AdvanceWidth, 216
Bottom, 216
Height, 216
LeftSideBearing, 216
RightSideBearing, 216
Top, 217
Width, 217
VectSharp.Font.FontUnderline, 257
FollowItalicAngle, 258
LineCap, 258
Position, 258
SkipDescenders, 258
Thickness, 258
VectSharp.FontFamily, 245
DefaultFontLibrary, 250
FamilyName, 250
FileName, 251
FontFamily, 247
IsBold, 251
IsItalic, 251
IsOblique, 251
IsStandardFamily, 251
ResolveFontFamily, 248, 249
StandardFamilies, 250
StandardFontFamilies, 246
StandardFontFamilyResources, 250
TrueTypeFile, 252
VectSharp.FontFamilyCreationException, 252
FontFamily, 253
FontFamilyCreationException, 253
VectSharp.FontLibrary, 254
ResolveFontFamily, 255, 256
VectSharp.Fonts, 28
VectSharp.Fonts.Nimbus, 532
Library, 532
VectSharp.Fonts.Nimbus/Nimbus.cs, 892
VectSharp.FormattedText, 261
Brush, 264
Font, 264
Format, 262, 263
FormattedText, 262
Script, 265
Spacing, 265
Text, 265
VectSharp.FormattedTextExtensions, 265
GetText, 266
Measure, 266
VectSharp.Frame, 267
Duration, 268
Frame, 267
Graphics, 268
VectSharp.GradientBrush, 281
GradientStops, 281
VectSharp.Gradients, 282
Cividis, 287
CividisColouring, 284
Inferno, 287
InfernoColouring, 284
Magma, 287
MagmaColouring, 285
Mako, 288
MakoColouring, 285
MutedRainbow, 288
MutedRainbowDiscrete, 288
OkabeItoRainbow, 288
OkabeItoRainbowDiscrete, 289
Plasma, 289
PlasmaColouring, 285
Rainbow, 289
RedToGreen, 289
RedYellowGreen, 290
Rocket, 290
RocketColouring, 286
TransparentToBlack, 290

Turbo, 290
TurboColouring, 286
Viridis, 291
ViridisColouring, 286
WhiteToBlack, 291
VectSharp.GradientStop, 291
Colour, 293
GradientStop, 292
MultiplyOpacity, 292
Offset, 293
VectSharp.GradientStops, 293
Count, 296
GetColourAt, 295
GetEnumerator, 295
GradientStops, 294, 295
operator Func< double, Colour >, 295
StopTolerance, 296
this[int index], 296
VectSharp.Graphics, 297
CopyToIGraphicsContext, 301
Crop, 301
DefaultFillRule, 335
DrawGraphics, 302–304
DrawRasterImage, 304–306
FillPath, 307
FillRectangle, 307, 308
FillText, 308, 310, 312, 313
FillTextOnPath, 313
FillTextUnderline, 315, 317
GetBounds, 318
GetTags, 318
Linearise, 318
MeasureText, 319
RasterisationMethod, 334
Restore, 320
Rotate, 320
RotateAt, 320
Save, 321
Scale, 321
SetClippingPath, 321, 322
StrokePath, 322
StrokeRectangle, 323, 324
StrokeText, 324–327
StrokeTextOnPath, 328
StrokeTextUnderline, 329–331
Transform, 331, 332
Translate, 333
TryRasterise, 334
UnbalancedStackAction, 335
UseUniqueTags, 335
VectSharp.GraphicsPath, 337
AddPath, 340
AddSmoothSpline, 340
AddText, 340, 342
AddTextOnPath, 342
AddTextUnderline, 343
Arc, 343, 344
Close, 344
ContainsPoint, 345
CubicBezierTo, 345, 346
Discretise, 346
EllipticalArc, 347
Flatten, 347
GetBounds, 348
GetFigures, 348
GetLinearisationPointsNormals, 348
GetNormalAtAbsolute, 349
GetNormalAtRelative, 349
GetPointAtAbsolute, 349
GetPointAtRelative, 350
GetPoints, 350
GetStroke, 350
GetTangentAtAbsolute, 351
GetTangentAtRelative, 351
Linearise, 352
LineTo, 352, 353
MeasureLength, 353
MoveTo, 353, 354
QuadraticBezierTo, 354
Reverse, 355
Segments, 356
Transform, 355
Triangulate, 356
VectSharp.IEasing, 370
Ease, 370
VectSharp.IFontLibrary, 375
ResolveFontFamily, 376, 377
VectSharp.IGraphicsContext, 378
CubicBezierTo, 380
DrawFilteredGraphics, 380
DrawRasterImage, 380
Fill, 381
FillStyle, 386
FillText, 381
Font, 386
Height, 386
LineCap, 386
LineJoin, 387
LineTo, 382
LineWidth, 387
MoveTo, 382
Rectangle, 382
Rotate, 383
Scale, 383
SetFillStyle, 383, 384
SetLineDash, 384
SetStrokeStyle, 384
StrokeStyle, 387
StrokeText, 385
Tag, 387
TextBaseline, 387
Transform, 385
Translate, 386
Width, 388
VectSharp.ImageSharpUtils, 28
VectSharp.ImageSharpUtils.ImageURIParser, 398

Parser, 399
VectSharp.ImageSharpUtils.RasterImageFile, 661
RasterImageFile, 662
VectSharp.ImageSharpUtils.RasterImageStream, 663
RasterImageStream, 664, 665
VectSharp.ImageSharpUtils/ImageUriParser.cs, 965
VectSharp.ImageSharpUtils/RasterImages.cs, 974
VectSharp.LinearEasing, 426
Ease, 427
LinearEasing, 426
VectSharp.LinearGradientBrush, 427
EndPoint, 430
LinearGradientBrush, 428, 429
MultiplyOpacity, 429
RelativeTo, 430
StartPoint, 430
VectSharp.LineDash, 436
DashArray, 438
LineDash, 437
Phase, 438
SolidLine, 438
UnitsOff, 438
UnitsOn, 438
VectSharp.Markdown, 29
VectSharp.Markdown.BlockRenderedEventArgs, 77
Block, 78
BlockRenderedEventArgs, 78
Graphics, 78
MarkdownContext, 79
VectSharp.Markdown.BlockRenderingEventArgs, 79
Block, 80
BlockRenderingEventArgs, 80
Graphics, 80
MarkdownContext, 81
VectSharp.Markdown.FormattedString, 259
Colour, 260
FormattedString, 259
IsBold, 260
IsItalic, 260
Text, 260
VectSharp.Markdown.HTTPUtils, 360
LogDownloads, 362
path, 361
ResolveEmojiUsingOpenMoji, 361
ResolveImageURI, 361
VectSharp.Markdown.InlineRenderedEventArgs, 402
Graphics, 403
Inline, 403
InlineRenderedEventArgs, 403
MarkdownContext, 403
VectSharp.Markdown.InlineRenderingEventArgs, 404
Graphics, 405
Inline, 405
InlineRenderingEventArgs, 405
MarkdownContext, 405
VectSharp.Markdown.Line, 414
Fragments, 414
InitialAscent, 415
Line, 414
VectSharp.Markdown.LineEventArgs, 439
Baseline, 441
ContentMaxX, 441
ContentMaxY, 441
ContentMinY, 441
Graphics, 441
Line, 441
LineEventArgs, 440
MarkdownContext, 442
PageMaxX, 442
PageMaxY, 442
VectSharp.Markdown.LineFragment, 443
GetAscent, 443
GetMaxX, 444
GetMaxY, 444
Render, 444
Tag, 445
Translate, 445
VectSharp.Markdown.LineMeasuredEventArgs, 446
Baseline, 447
ContentMaxX, 447
ContentMaxY, 448
ContentMinY, 448
Graphics, 448
Line, 448
LineMeasuredEventArgs, 447
MarkdownContext, 448
PageMaxX, 449
PageMaxY, 449
VectSharp.Markdown.Margins, 479
Bottom, 480
Left, 480
Margins, 479
Right, 480
Top, 480
VectSharp.Markdown.MarkdownContext, 486
BottomRight, 490
Clone, 487
Colour, 490
CurrentLine, 490
CurrentPage, 490
Cursor, 490
Font, 490
ForbiddenAreasLeft, 491
ForbiddenAreasRight, 491
GetMaxX, 487, 488
GetMinX, 488, 489
Headings, 491
InternalAnchors, 491
LinkDestinations, 492
ListDepth, 492
MarginBottomRight, 492
StartNewPage, 489
StrikeThrough, 492
Tag, 492
Translation, 493
Underline, 493

VectSharp.Markdown.MarkdownEmojiExtension, 493
UseUnicodeEmoji, 493
VectSharp.Markdown.MarkdownRenderer, 494
AllowPageBreak, 501
BackgroundColour, 501
BaseFontSize, 501
BaseImageUri, 502
BaseLinkUri, 502
BlockRendered, 515
BlockRendering, 515
BoldFontFamily, 502
BoldItalicFontFamily, 502
BoldUnderlineThickness, 502
Bullets, 503
CodeBlockBackgroundColour, 503
CodeFont, 503
CodeFontBold, 503
CodeFontBoldItalic, 504
CodeFontItalic, 504
CodeInlineBackgroundColour, 504
CodeInlineMargin, 504
EmojiUriResolver, 504
EmojiURLMapping, 505
ForegroundColour, 505
HeaderFontSizeMultipliers, 505
HeaderLineColour, 505
HeaderLineThicknesses, 506
ImageMarginTolerance, 506
ImageMultiplier, 506
ImageSideMargin, 506
ImageUnitMultiplier, 506
ImageUriResolver, 507
IndentWidth, 507
InlineRendered, 515
InlineRendering, 515
InsertedColour, 507
ItalicFontFamily, 507
LineMeasured, 516
LineRendered, 516
LineRendering, 516
LinkColour, 508
LinkUriResolver, 508
Margins, 508
MarkdownPipelineBuilder, 508
MarkdownRenderer, 499
MarkedColour, 508
MathFontScalingFactor, 509
PageFinished, 516
PageSize, 509
PageStarted, 516
QuoteBlockBackgroundColour, 509
QuoteBlockBarColour, 509
QuoteBlockBarWidth, 509
QuoteBlockIndentWidth, 510
RasterImageLoader, 510
RegularFontFamily, 510
Render, 499
RenderSinglePage, 500
SpaceAfterHeading, 510
SpaceAfterLine, 510
SpaceAfterParagraph, 511
SpaceBeforeHeading, 511
SpaceBeforeParagaph, 511
SubscriptShift, 511
SubSuperscriptFontSize, 511
SuperscriptShift, 512
SyntaxHighlighter, 512
TableCellMargins, 512
TableHeaderRowSeparatorColour, 512
TableHeaderRowSeparatorThickness, 513
TableHeaderSeparatorThickness, 513
TableRowSeparatorColour, 513
TableVAlign, 513
TaskListCheckedBullet, 513
TaskListUncheckedBullet, 514
ThematicBreakLineColour, 514
ThematicBreakThickness, 514
UnderlineThickness, 515
VerticalAlignment, 498
VectSharp.Markdown.PageFinishedEventArgs, 576
Graphics, 577
MarkdownContext, 577
Page, 577
PageFinishedEventArgs, 577
VectSharp.Markdown.PageStartedEventArgs, 578
Graphics, 579
MarkdownContext, 579
Page, 579
PageStartedEventArgs, 579
VectSharp.Markdown.RectangleFragment, 674
Colour, 677
GetAscent, 676
GetMaxX, 676
GetMaxY, 676
RectangleFragment, 675
Render, 676
Size, 677
TopLeft, 678
Translate, 677
VectSharp.Markdown.SyntaxHighlighter, 765
GetSyntaxHighlightedLines, 765
VectSharp.Markdown.TextFragment, 766
Colour, 770
Font, 770
GetAscent, 768
GetMaxX, 768
GetMaxY, 768
Position, 770
Render, 769
Text, 770
TextFragment, 767
Translate, 769
VectSharp.Markdown.UnderlineFragment, 798
Colour, 802
End, 802
GetAscent, 800

GetMaxX, 801
GetMaxY, 801
Render, 801
Start, 803
Thickness, 803
Translate, 802
UnderlineFragment, 800
VectSharp.Markdown/HtmlTag.cs, 893
VectSharp.Markdown/Line.cs, 899
VectSharp.Markdown/MarkdownContext.cs, 905
VectSharp.Markdown/MarkdownRenderer.Blocks.cs, 909
VectSharp.Markdown/MarkdownRenderer.cs, 919
VectSharp.Markdown/MarkdownRenderer.Events.cs, 926
VectSharp.Markdown/MarkdownRenderer.Inlines.cs, 933
VectSharp.Markdown/MarkdownRenderer.Properties.cs, 943
VectSharp.Markdown/MarkdownRenderer.Tables.cs, 949
VectSharp.Markdown/SyntaxHighlighting.cs, 953
VectSharp.Markdown/UnicodeEmojiParser.cs, 956
VectSharp.Markdown/Word.cs, 958
VectSharp.MarkdownCanvas, 29
VectSharp.MarkdownCanvas.MarkdownCanvasControl, 481
Document, 484
DocumentProperty, 482
DocumentSource, 484
DocumentSourceProperty, 483
MarkdownCanvasControl, 482
MaxRenderWidth, 484
MaxRenderWidthProperty, 483
MinRenderWidth, 485
MinRenderWidthProperty, 483
MinVariation, 485
MinVariationProperty, 483
Renderer, 485
TextConversionOption, 485
TextConversionOptionsProperty, 484
VectSharp.MarkdownCanvas/ImageCache.cs, 960
VectSharp.MarkdownCanvas/MarkdownCanvas.axaml.cs, 961
VectSharp.MultiFontLibrary, 529
MultiFontLibrary, 530
ResolveFontFamily, 531
VectSharp.MuPDFUtils, 30
VectSharp.MuPDFUtils.ImageURIParser, 399
Parser, 400
VectSharp.MuPDFUtils.PDFParser, 601
FromFile, 601–603
FromStream, 603–605
VectSharp.MuPDFUtils.RasterImageFile, 662
RasterImageFile, 663
VectSharp.MuPDFUtils.RasterImageStream, 665
RasterImageStream, 666, 667
VectSharp.MuPDFUtils/ImageURIParser.cs, 966
VectSharp.MuPDFUtils/PDFParser.cs, 969
VectSharp.MuPDFUtils/RasterImages.cs, 977
VectSharp.Page, 573
Background, 575
Crop, 574, 575
Graphics, 575
Height, 575
Page, 573
Width, 575
VectSharp.PDF, 30
VectSharp.PDF.AnnotationStyle, 50
AnnotationStyle, 51
BorderColour, 51
BorderDash, 51
BorderWidth, 51
VectSharp.PDF.AnnotationStyleCollection, 52
Add, 53
AnnotationStyleCollection, 53
Clear, 53
Contains, 54
ContainsKey, 54
CopyTo, 54
Count, 55
DefaultStyle, 55
GetEnumerator, 54
IsReadOnly, 55
Keys, 55
Remove, 54
Styles, 56
this[string key], 56
TryGetValue, 55
Values, 56
VectSharp.PDF.Figures, 30
VectSharp.PDF.OptionalContentGroups, 30
OptionalContentGroupBooleanOperation, 31
VectSharp.PDF.OptionalContentGroups.OptionalContentGroup, 540
CreateSetOCGLink, 542
DefaultExportState, 543
DefaultPrintState, 543
DefaultState, 543
DefaultViewState, 543
Intent, 543
Intents, 541
Name, 544
Operation, 544
OptionalContentGroup, 541
ToString, 542
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupAndExpres 544
Operation, 545
OptionalContentGroupAndExpression, 545
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupExpression 546
Arguments, 549
Operation, 549
operator false, 547
operator true, 547

operator!, 548
operator&, 548
operator|, 548
ToString, 549
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupNotExpression, 550
Operation, 551
OptionalContentGroupNotExpression, 551
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupOrExpression, 551
Operation, 553
OptionalContentGroupOrExpression, 552
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupSettings, 553
Groups, 554
OptionalContentGroupSettings, 554
OptionalContentGroupTree, 554
RadioButtonGroups, 554
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeLabel, 555
LabelOptionalContentGroup, 557
LabelString, 558
LabelType, 558
operator OptionalContentGroupTreeLabel, 557
OptionalContentGroupTreeLabel, 556
Type, 556
VectSharp.PDF.OptionalContentGroups.OptionalContentGroupTreeNode, 558
Add, 560
Children, 562
Clear, 560
Contains, 560
CopyTo, 560
Count, 562
GetEnumerator, 560
IndexOf, 561
Insert, 561
IsReadOnly, 563
Label, 563
operator OptionalContentGroupTreeNode, 561, 562
OptionalContentGroupTreeNode, 559
Remove, 562
RemoveAt, 562
this[int index], 563
VectSharp.PDF.OutlineTree, 563
Add, 564
Clear, 565
Contains, 565
CopyTo, 565
Count, 567
CreateFromHeadings, 565
GetEnumerator, 566
IndexOf, 566
Insert, 566
IsReadOnly, 567
OutlineTree, 564
Remove, 566
RemoveAt, 566
this[int index], 567
TopLevelItems, 567
VectSharp.PDF.OutlineTreeNode, 568
AddExpression, 569
Bold, 571
Children, 571
Clear, 569
CreateExpression, 571
Contains, 569
CopyTo, 570
Count, 571
DestinationTag, 572
GetEnumerator, 570
IndexOf, 570
Insert, 570
IsReadOnly, 572
Italic, 572
OutlineTreeNode, 569
Remove, 570
RemoveAt, 571
this[int index], 572
Title, 572
VectSharp.PDF.PDFContextInterpreter, 590
CreatePDFDocument, 591
SaveAsPDF, 592
TextOptions, 591
VectSharp.PDF.PDFContextInterpreter.FilterOption, 228
Default, 229
FilterOperations, 229
FilterOption, 229
Operation, 230
RasterisationResolution, 230
RasterisationResolutionRelative, 230
VectSharp.PDF.PDFDocument, 593
Catalog, 595
Contents, 595
DefaultEncoding, 595
Info, 596
Pages, 596
PDFDocument, 594
PDFVersion, 596
Write, 594, 595
VectSharp.PDF.PDFMetadata, 596
Author, 598
CreationDate, 598
CreationDateTimeZone, 598
Creator, 599
CustomProperties, 599
ExcludeMetadata, 599
Keywords, 599
ModificationDate, 599
ModificationDateTimeZone, 600
NoMetadata, 598
PDFMetadata, 597
Producer, 600
Subject, 600
Title, 600

VectSharp.PDF/AnnotationStyles.cs, 979
VectSharp.PDF/Figures.cs, 981
VectSharp.PDF/OptionalContentGroups.cs, 986
VectSharp.PDF/OutlineTree.cs, 993
VectSharp.PDF/PDFContext.cs, 996
VectSharp.PDF/PDFContextInterpreter.cs, 1006
VectSharp.PDF/PDFDocument.cs, 1018
VectSharp.PDF/PDFMetadata.cs, 1022
VectSharp.PDF/PDFProcessFigure.cs, 1024
VectSharp.Plots, 31
VectSharp.Plots.ActionDataPointElement, 35
ActionDataPointElement, 36
Plot, 36
PlotAction, 37
VectSharp.Plots.Area< T >, 56
Area, 57
CoordinateSystem, 59
Data, 59
GetBaseline, 60
Plot, 59
PresentationAttributes, 60
Smooth, 60
Tag, 60
VectSharp.Plots.Bars< T >, 70
Bars, 71–74
CoordinateSystem, 75
Data, 75
GetBaseline, 75
Margin, 75
Plot, 74
PresentationAttributes, 75
Tag, 76
VectSharp.Plots.BoxPlot, 84
Box1, 87
Box2, 87
BoxPlot, 85
BoxPresentationAttributes, 87
CentreSymbol, 87
CentreSymbolPresentationAttributes, 87
CoordinateSystem, 88
Direction, 88
NotchSize, 88
NotchWidth, 88
Plot, 86
Position, 88
Tag, 89
Whisker1, 89
Whisker2, 89
WhiskersPresentationAttributes, 89
WhiskerWidth, 89
Width, 90
VectSharp.Plots.CategoricalBars< T >, 92
CategoricalBars, 93, 94
VectSharp.Plots.CategoricalCoordinateSystem1D< T >, 94
CategoricalCoordinateSystem1D, 96
Coordinates, 97
ToPlotCoordinates, 96
VectSharp.Plots.ClusteredBars, 99
ClusteredBars, 100, 101
CoordinateSystem, 102
Data, 102
GetBaseline, 102
InterClusterMargin, 103
IntraClusterMargin, 103
Plot, 102
PresentationAttributes, 103
Tag, 103
Vertical, 103
VectSharp.Plots.CompositeCoordinateSystem2D< T1, T2 >, 172
CompositeCoordinateSystem2D, 173
CoordinateSystemX, 173
CoordinateSystemY, 174
ToPlotCoordinates, 173
VectSharp.Plots.ContinuousAxis, 180
ArrowSize, 182
ContinuousAxis, 181
CoordinateSystem, 182
EndPoint, 182
Plot, 181
PresentationAttributes, 182
StartPoint, 182
Tag, 183
VectSharp.Plots.ContinuousAxisLabels, 183
Alignment, 185
Baseline, 185
ContinuousAxisLabels, 184
CoordinateSystem, 185
EndPoint, 186
IntervalCount, 186
Plot, 185
Position, 186
PresentationAttributes, 186
Rotation, 186
StartPoint, 187
Tag, 187
TextFormat, 187
VectSharp.Plots.ContinuousAxisTicks, 188
ContinuousAxisTicks, 189
CoordinateSystem, 190
EndPoint, 190
IntervalCount, 190
Plot, 189
PresentationAttributes, 190
SizeAbove, 191
SizeBelow, 191
StartPoint, 191
Tag, 191
VectSharp.Plots.ContinuousAxisTitle, 192
Alignment, 195
Baseline, 195
ContinuousAxisTitle, 193, 194
CoordinateSystem, 195
EndPoint, 195
FollowAxis, 195

Plot, 194
Position, 196
PresentationAttributes, 196
Rotation, 196
StartPoint, 196
Tag, 196
Title, 197
VectSharp.Plots.CoordinateSystem< T >, 201
CoordinateFunction, 203
CoordinateSystem, 202
ToPlotCoordinates, 202
VectSharp.Plots.CoordinateSystem1D< T >, 203
CoordinateFunction, 205
CoordinateSystem1D, 204
ToPlotCoordinates, 205
VectSharp.Plots.DataLabels< T >, 206
Alignment, 208
Baseline, 208
CoordinateSystem, 208
Data, 208
DataLabels, 207
Label, 209
Margin, 209
Plot, 207
PresentationAttributes, 209
Rotation, 209
Tag, 210
VectSharp.Plots.DataLine< T >, 210
CoordinateSystem, 212
Data, 212
DataLine, 211
Plot, 212
PresentationAttributes, 212
Smooth, 213
Tag, 213
VectSharp.Plots.ExponentialTrendLine, 220
CoordinateSystem, 223
ExponentialTrendLine, 221, 222
Intercept, 223
MaxX, 223
MaxY, 224
MinX, 224
MinY, 224
Plot, 223
PresentationAttributes, 224
Slope, 224
Tag, 225
VectSharp.Plots.Function2D, 268
Colouring, 271
CoordinateSystem, 271
Function, 271
Function2D, 270
Plot, 270
PlotType, 269
RasterResolutionX, 271
RasterResolutionY, 271
SampledPointElement, 272
Tag, 272
Type, 272
VectSharp.Plots.Function2DGrid, 272
DataPoints, 275
Function2DGrid, 274
GridType, 273
MaxX, 275
MaxY, 276
MaxZ, 276
MinX, 276
MinY, 276
MinZ, 276
StepsX, 277
StepsY, 277
ToRectangular, 275
Type, 277
VectSharp.Plots.GraphicsDataPointElement, 335
Graphics, 337
GraphicsDataPointElement, 336
Plot, 336
VectSharp.Plots.Grid, 357
CoordinateSystem, 359
Grid, 358
IntervalCount, 359
Plot, 358
PresentationAttributes, 359
Side1End, 359
Side1Start, 359
Side2End, 360
Side2Start, 360
Tag, 360
VectSharp.Plots.IContinuousCoordinateSystem, 362
GetAround, 363
IsDirectionStraight, 363
IsLinear, 364
Resolution, 364
VectSharp.Plots.IContinuousInvertibleCoordinateSystem, 365
ToDataCoordinates, 365
VectSharp.Plots.ICoordinateSystem< T >, 366
ToPlotCoordinates, 367
VectSharp.Plots.ICoordinateSystem1D< T >, 367
ToPlotCoordinates, 368
VectSharp.Plots.IDataPointElement, 368
Plot, 369
VectSharp.Plots.IPlotElement, 406
CoordinateSystem, 408
Plot, 408
VectSharp.Plots.LinearCoordinateSystem1D, 415
LinearCoordinateSystem1D, 416
Max, 417
Min, 417
Scale, 417
ToPlotCoordinates, 417
VectSharp.Plots.LinearCoordinateSystem2D, 418
GetAround, 421
IsDirectionStraight, 421
IsLinear, 424
LinearCoordinateSystem2D, 420, 421

MaxX, 424
MaxY, 424
MinX, 424
MinY, 425
Resolution, 425
ScaleX, 425
ScaleY, 425
ToDataCoordinates, 422
ToPlotCoordinates, 422
VectSharp.Plots.LinearTrendLine, 431
CoordinateSystem, 434
Intercept, 434
LinearTrendLine, 432, 433
MaxX, 434
MaxY, 434
MinX, 435
MinY, 435
Plot, 433
PresentationAttributes, 435
Slope, 435
Tag, 435
VectSharp.Plots.LinLogCoordinateSystem2D, 449
GetAround, 453
IsDirectionStraight, 453
IsLinear, 455
LinLogCoordinateSystem2D, 451, 452
MaxX, 455
MaxY, 455
MinX, 455
MinY, 456
Resolution, 456
ScaleX, 456
ScaleY, 456
ToDataCoordinates, 454
ToPlotCoordinates, 454
VectSharp.Plots.LogarithmicCoordinateSystem1D, 457
LogarithmicCoordinateSystem1D, 457, 458
Max, 458
Min, 459
Scale, 459
ToPlotCoordinates, 458
VectSharp.Plots.LogarithmicCoordinateSystem2D, 459
GetAround, 463
IsDirectionStraight, 463
IsLinear, 465
LogarithmicCoordinateSystem2D, 461, 462
MaxX, 465
MaxY, 465
MinX, 465
MinY, 466
Resolution, 466
ScaleX, 466
ScaleY, 466
ToDataCoordinates, 464
ToPlotCoordinates, 464
VectSharp.Plots.LogarithmicTrendLine, 467
CoordinateSystem, 470
Intercept, 470
LogarithmicTrendLine, 468, 469
MaxX, 470
MaxY, 470
MinX, 471
MinY, 471
Plot, 469
PresentationAttributes, 471
Slope, 471
Tag, 471
VectSharp.Plots.LogLinCoordinateSystem2D, 472
GetAround, 475
IsDirectionStraight, 475
IsLinear, 477
LogLinCoordinateSystem2D, 474, 475
MaxX, 477
MaxY, 477
MinX, 477
MinY, 478
Resolution, 478
ScaleX, 478
ScaleY, 478
ToDataCoordinates, 476
ToPlotCoordinates, 476
VectSharp.Plots.MovingAverageTrendLine, 525
CoordinateSystem, 527
Data, 528
MovingAverageTrendLine, 526, 527
Period, 528
Plot, 527
PresentationAttributes, 528
Tag, 528
Weight, 528
VectSharp.Plots.PathDataPointElement, 588
Path, 590
PathDataPointElement, 589
Plot, 589
VectSharp.Plots.Pie, 609
Centre, 612
Clockwise, 612
CoordinateSystem, 612
Data, 612
InnerRadius, 613
OuterRadius, 613
Pie, 610, 611
Plot, 611
PresentationAttributes, 613
StartAngle, 613
Tag, 613
VectSharp.Plots.Plot, 614
AddPlotElement, 615
AddPlotElements, 616
GetAll< T >, 616
GetFirst< T >, 617
NormalisationMode, 615
Plot, 615
PlotElements, 618
RemovePlotElement, 617
Render, 617, 618

WhiskerType, 615
VectSharp.Plots.PlotElement< T >, 619
CoordinateSystem, 620
Plot, 620
PlotAction, 621
PlotElement, 620
VectSharp.Plots.PlotElementPresentationAttributes, 621
Fill, 622
Font, 622
LineCap, 623
LineDash, 623
LineJoin, 623
LineWidth, 623
PlotElementPresentationAttributes, 622
Stroke, 623
VectSharp.Plots.PolynomialTrendLine, 637
Coefficients, 640
CoordinateSystem, 640
MaxX, 640
MaxY, 641
MinX, 641
MinY, 641
Plot, 640
PolynomialTrendLine, 638, 639
PresentationAttributes, 641
Tag, 641
VectSharp.Plots.PowerLawTrendLine, 642
CoordinateSystem, 645
Intercept, 645
MaxX, 645
MaxY, 645
MinX, 646
MinY, 646
Plot, 644
PowerLawTrendLine, 643, 644
PresentationAttributes, 646
Slope, 646
Tag, 646
VectSharp.Plots.ScatterPoints< T >, 687
CoordinateSystem, 689
Data, 689
DataPointElement, 689
Plot, 689
PresentationAttributes, 690
ScatterPoints, 688
Size, 690
Tag, 690
VectSharp.Plots.StackedBars, 747
CoordinateSystem, 750
Data, 751
GetBaseline, 751
Margin, 751
Plot, 750
PresentationAttributes, 751
StackedBars, 748, 749
Tag, 751
Vertical, 752
VectSharp.Plots.Swarm, 761
CoordinateSystem, 763
Data, 763
Direction, 763
Plot, 762
PointMargin, 763
PointSize, 764
Position, 764
PresentationAttributes, 764
Swarm, 762
SwarmPointElement, 764
Tag, 764
VectSharp.Plots.TextLabel< T >, 771
Alignment, 773
Baseline, 773
CoordinateSystem, 773
Label, 773
Plot, 772
Position, 773
PresentationAttributes, 774
Rotation, 774
Tag, 774
TextLabel, 772
VectSharp.Plots.Violin, 805
CoordinateSystem, 807
Data, 808
Direction, 808
MaxBins, 808
Plot, 807
Position, 808
PresentationAttributes, 808
Sides, 809
Smooth, 809
Tag, 809
Violin, 807
ViolinSide, 806
Width, 809
VectSharp.Plots/Axes.cs, 1032
VectSharp.Plots/Bars.cs, 1044
VectSharp.Plots/BoxPlot.cs, 1058
VectSharp.Plots/CoordinateSystems.cs, 1062
VectSharp.Plots/DataPoints.cs, 1076
VectSharp.Plots/Function2D.cs, 1085
VectSharp.Plots/Pie.cs, 1097
VectSharp.Plots/Plot.Area.cs, 1099
VectSharp.Plots/Plot.BarChart.cs, 1106
VectSharp.Plots/Plot.BoxPlot.cs, 1121
VectSharp.Plots/Plot.cs, 1129
VectSharp.Plots/Plot.Function1D.cs, 1133
VectSharp.Plots/Plot.Function2D.cs, 1135
VectSharp.Plots/Plot.Histogram.cs, 1140
VectSharp.Plots/Plot.Lines.cs, 1162
VectSharp.Plots/Plot.Pie.cs, 1170
VectSharp.Plots/Plot.ScatterPlot.cs, 1175
VectSharp.Plots/Plot.SwarmPlot.cs, 1180
VectSharp.Plots/Plot.ViolinPlot.cs, 1188
VectSharp.Plots/Swarm.cs, 1196
VectSharp.Plots/Trendlines.cs, 1198

VectSharp.Plots/Violin.cs, 1213
VectSharp.Point, 624
Bounds, 627
Count, 633
GetEnumerator, 627
IsEqual, 628
Max, 628
Min, 628
Modulus, 629
Normalize, 629
operator, 629
operator double[], 630
operator Point, 630
operator∗, 631
operator+, 631
operator-, 632
Point, 625
this[int index], 633
X, 632
Y, 632
VectSharp.RadialGradientBrush, 647
Centre, 649
FocalPoint, 650
MultiplyOpacity, 649
RadialGradientBrush, 648
Radius, 650
VectSharp.Raster, 33
VectSharp.Raster.ImageSharp, 33
OutputFormats, 34
VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter, 391
Rasterise, 392
SaveAsAnimatedGIF, 393, 394
SaveAsAnimatedPNG, 395
SaveAsImage, 396, 397
SaveAsRawBytes, 397, 398
VectSharp.Raster.ImageSharp.ImageSharpContextInterpreter.UnknownFormatException, 803
Format, 804
VectSharp.Raster.ImageSharp/GradientBrushApplicator.cs, 1217
VectSharp.Raster.ImageSharp/ImageSharpContext.cs, 1220
VectSharp.Raster.Raster, 651
Rasterise, 652
SaveAsAnimatedPNG, 653
SaveAsJPEG, 654
SaveAsPNG, 654, 655
SaveAsRawBytes, 655
VectSharp.Raster/Raster.cs, 1240
VectSharp.RasterImage, 656
ClearPNGCache, 659
DataHolder, 659
Dispose, 659
HasAlpha, 659
Height, 659
Id, 660
ImageDataAddress, 660
Interpolate, 660
PNGStream, 660
RasterImage, 657, 658
Width, 660
VectSharp.Rectangle, 667
Centre, 673
Intersection, 670
IsNaN, 671
Location, 672
NaN, 673
Rectangle, 669, 670
Size, 673
Union, 671, 672
VectSharp.ResourceFontFamily, 686
ResourceFontFamily, 687
ResourceName, 687
VectSharp.Segment, 694
Clone, 695
Flatten, 695
FlattenForOffsetAndGetTangents, 696
GetLinearisationTangents, 696
GetPointAt, 696
GetTangentAt, 697
Linearise, 697
Measure, 698
Point, 698
Points, 698
Transform, 698
Type, 699
VectSharp.SimpleFontLibrary, 699
Add, 703, 704
ResolveFontFamily, 704
SimpleFontLibrary, 700–702
VectSharp.Size, 705
Height, 706
Size, 705
Width, 706
VectSharp.SolidColourBrush, 737
A, 739
B, 739
Colour, 739
G, 740
MultiplyOpacity, 738
operator SolidColourBrush, 739
R, 740
SolidColourBrush, 738
VectSharp.SplineEasing, 740
ControlPoint1, 742
ControlPoint2, 742
Ease, 741
SplineEasing, 741
VectSharp.SVG, 34
VectSharp.SVG.Parser, 585
FromFile, 585
FromStream, 586
FromString, 586
ParseImageURI, 587
ParseSVGURI, 586

VectSharp.SVG.SVGContextInterpreter, 752
SaveAsAnimatedSVG, 754, 755
SaveAsAnimatedSVGWithFrames, 756, 757
SaveAsSVG, 758, 759
TextOptions, 753
WriteSVGXML, 759, 760
VectSharp.SVG.SVGContextInterpreter.FilterOption, 231
Default, 232
FilterOperations, 231
FilterOption, 232
Operation, 232
RasterisationResolution, 233
RasterisationResolutionRelative, 233
VectSharp.SVG/SVGContext.cs, 1244
VectSharp.SVG/SVGParser.cs, 1299
VectSharp.TextSpacing, 775
Default, 778
Equals, 777
GetHashCode, 777
NonWhitespaceAdd, 778
NonWhitespaceScale, 778
operator!=, 777
operator==, 777
TextSpacing, 776
WhitespaceAdd, 778
WhitespaceScale, 779
VectSharp.ThreeD, 34
VectSharp.ThreeD.AmbientLightSource, 37
AmbientLightSource, 38
CastsShadow, 39
GetLightAt, 39
GetObstruction, 39
Intensity, 40
VectSharp.ThreeD.AreaLightSource, 61
AreaLightSource, 62
CastsShadow, 64
Center, 64
Direction, 64
DistanceAttenuationExponent, 64
GetLightAt, 63
GetObstruction, 63
Intensity, 64
PenumbraAttenuationExponent, 65
PenumbraRadius, 65
Radius, 65
ShadowSamplingPointCount, 65
SourceDistance, 65
VectSharp.ThreeD.ColourMaterial, 118
Colour, 120
ColourMaterial, 119
GetColour, 119
VectSharp.ThreeD.ILightSource, 388
CastsShadow, 390
GetLightAt, 389
GetObstruction, 389
VectSharp.ThreeD.IMaterial, 400
GetColour, 401
VectSharp.ThreeD.IScene, 409
AddElement, 410
AddRange, 410
Replace, 410, 411
SceneElements, 411
SceneLock, 411
VectSharp.ThreeD.LightIntensity, 412
Deconstruct, 413
Direction, 413
Intensity, 413
LightIntensity, 412
VectSharp.ThreeD.MaskedLightSource, 517
AngleAttenuationExponent, 520
CastsShadow, 521
Direction, 521
Distance, 521
DistanceAttenuationExponent, 521
GetLightAt, 520
GetObstruction, 520
Intensity, 521
MaskedLightSource, 518, 519
Origin, 522
Position, 522
VectSharp.ThreeD.ObjectFactory, 533
CreateCube, 533
CreateCuboid, 534
CreatePoints, 535
CreatePolygon, 535
CreatePrism, 536
CreateRectangle, 537
CreateSphere, 538
CreateTetrahedron, 538
CreateWireframe, 539
VectSharp.ThreeD.ParallelLightSource, 581
CastsShadow, 584
Direction, 584
GetLightAt, 583
GetObstruction, 583
Intensity, 584
ParallelLightSource, 582
ReverseDirection, 584
VectSharp.ThreeD.PhongMaterial, 606
AmbientReflectionCoefficient, 608
Colour, 608
DiffuseReflectionCoefficient, 608
GetColour, 607
PhongMaterial, 607
SpecularReflectionCoefficient, 609
SpecularShininess, 609
VectSharp.ThreeD.PointLightSource, 633
CastsShadow, 636
DistanceAttenuationExponent, 636
GetLightAt, 635
GetObstruction, 635
Intensity, 636
PointLightSource, 634
Position, 636
VectSharp.ThreeD.Scene, 691

AddElement, 692
AddRange, 692
Replace, 693
Scene, 692
SceneElements, 694
SceneLock, 694
VectSharp.ThreeD.SpotlightLightSource, 743
AngleAttenuationExponent, 745
BeamWidthAngle, 746
CastsShadow, 746
CutoffAngle, 746
Direction, 746
DistanceAttenuationExponent, 746
GetLightAt, 745
GetObstruction, 745
Intensity, 747
Position, 747
SpotlightLightSource, 744
VectSharp.ThreeD/Lights.cs, 1346
VectSharp.ThreeD/Materials.cs, 1357
VectSharp.ThreeD/ObjectFactory.cs, 1359
VectSharp.ThreeD/Scene.cs, 1365
VectSharp.Transition, 779
Duration, 781
Easings, 781
OverallEasing, 781
Transition, 780
VectSharp.TrueTypeFile, 782
Destroy, 784
FontStream, 795
Get1000EmAscent, 784
Get1000EmDescent, 784
Get1000EmGlyphBearings, 784
Get1000EmGlyphVerticalMetrics, 785
Get1000EmGlyphWidth, 785
Get1000EmKerning, 786
Get1000EmUnderlineIntersections, 787
Get1000EmUnderlinePosition, 787
Get1000EmUnderlineThickness, 787
Get1000EmWinAscent, 788
Get1000EmXMax, 788
Get1000EmXMin, 788
Get1000EmYMax, 788
Get1000EmYMin, 789
GetFirstCharIndex, 789
GetFontFamilyName, 789
GetFontName, 789
GetFullFontFamilyName, 790
GetGlyphIndex, 790
GetGlyphPath, 790, 791
GetItalicAngle, 791
GetLastCharIndex, 791
GetNames, 792
GetUnitsPerEm, 793
IsBold, 793
IsFixedPitch, 793
IsItalic, 793
IsOblique, 794
IsScript, 794
IsSerif, 794
SubsetFont, 794
VectSharp.TrueTypeFile.Bearings, 76
LeftSideBearing, 76
RightSideBearing, 77
VectSharp.TrueTypeFile.ClassDefinitionTable.ClassRangeRecord, 97
Class, 98
ClassRangeRecord, 98
EndGlyphID, 98
StartGlyphID, 98
VectSharp.TrueTypeFile.CoverageTable.RangeRecord, 650
EndGlyphID, 651
RangeRecord, 651
StartCoverageIndex, 651
StartGlyphID, 651
VectSharp.TrueTypeFile.PairKerning, 580
Glyph1Advance, 580
Glyph1Placement, 580
Glyph2Advance, 581
Glyph2Placement, 581
VectSharp.TrueTypeFile.TrueTypeName, 795
Name, 796
NameId, 796
NameIdentifier, 796
VectSharp.TrueTypeFile.TrueTypePoint, 796
IsOnCurve, 797
X, 797
Y, 797
VectSharp.TrueTypeFile.VerticalMetrics, 804
YMax, 805
YMin, 805
VectSharp.UnbalancedStackException, 798
VectSharp/Animation.cs, 1367
VectSharp/Brush.cs, 1388
VectSharp/Colour.cs, 1395
VectSharp/Document.cs, 1403
VectSharp/Enums.cs, 1405
VectSharp/Filters/BoxBlurFilter.cs, 1409
VectSharp/Filters/ColourMatrixFilter.cs, 1415
VectSharp/Filters/CompositeFilter.cs, 1423
VectSharp/Filters/ConvolutionFilter.cs, 1425
VectSharp/Filters/Filters.cs, 1429
VectSharp/Filters/GaussianBlurFilter.cs, 1432
VectSharp/Filters/MaskFilter.cs, 1436
VectSharp/Font.cs, 1438
VectSharp/FontLibrary.cs, 1447
VectSharp/FormattedText.cs, 1457
VectSharp/Gradients.cs, 1467
VectSharp/Graphics.cs, 1479
VectSharp/Graphics.Text.cs, 1501
VectSharp/GraphicsAction.cs, 1519
VectSharp/GraphicsPath.cs, 1523
VectSharp/ImageFormats.cs, 1580
VectSharp/Point.cs, 1594
VectSharp/RasterImage.cs, 1600

VectSharp/Segment.cs, 1604
VectSharp/SmoothSpline.cs, 1619
VectSharp/StandardColours.cs, 1620
VectSharp/TrueType.cs, 1629
Vertical
VectSharp.Plots.ClusteredBars, 103
VectSharp.Plots.StackedBars, 752
VerticalAlignment
VectSharp.Markdown.MarkdownRenderer, 498
Violet
VectSharp.Colours, 170
Violin
VectSharp.Plots.Violin, 807
ViolinSide
VectSharp.Plots.Violin, 806
Viridis
VectSharp.Gradients, 291
ViridisColouring
VectSharp.Gradients, 286

Weight
VectSharp.Plots.MovingAverageTrendLine, 528
Wheat
VectSharp.Colours, 171
Whisker1
VectSharp.Plots.BoxPlot, 89
Whisker2
VectSharp.Plots.BoxPlot, 89
WhiskersPresentationAttributes
VectSharp.Plots.BoxPlot, 89
WhiskerType
VectSharp.Plots.Plot, 615
WhiskerWidth
VectSharp.Plots.BoxPlot, 89
White
VectSharp.Colours, 171
WhiteSmoke
VectSharp.Colours, 171
WhitespaceAdd
VectSharp.TextSpacing, 778
WhitespaceScale
VectSharp.TextSpacing, 779
WhiteToBlack
VectSharp.Gradients, 291
Width
VectSharp.Animation, 50
VectSharp.Font.DetailedFontMetrics, 217
VectSharp.IGraphicsContext, 388
VectSharp.Page, 575
VectSharp.Plots.BoxPlot, 90
VectSharp.Plots.Violin, 809
VectSharp.RasterImage, 660
VectSharp.Size, 706
WinAscent
VectSharp.Font, 244
WithAlpha
VectSharp.Colour, 114, 115
VectSharp.Filters.ColourMatrix, 127
Write
VectSharp.PDF.PDFDocument, 594, 595
WriteSVGXML
VectSharp.SVG.SVGContextInterpreter, 759, 760

X
VectSharp.Colour, 117
VectSharp.Point, 632
VectSharp.TrueTypeFile.TrueTypePoint, 797

Y
VectSharp.Colour, 117
VectSharp.Point, 632
VectSharp.TrueTypeFile.TrueTypePoint, 797
Yellow
VectSharp.Colours, 171
YellowGreen
VectSharp.Colours, 171
YMax
VectSharp.Font, 244
VectSharp.TrueTypeFile.VerticalMetrics, 805
YMin
VectSharp.Font, 244
VectSharp.TrueTypeFile.VerticalMetrics, 805

ZIndex
VectSharp.Canvas.SKRenderAction, 726