MuPDFCore

1.8.0

Generated by Doxygen 1.9.5

1 MuPDFCore: Multiplatform .NET bindings for MuPDF 1
1.1 Getting started . . . 1
1.2 Usage . . . 2
1.2.1 Documentation . . . 2
1.2.2 Minimal working example . . . 2
1.2.3 Examples . . . 2
1.3 Building from source . . . 3
1.3.1 1. Building libmupdf . . . 3
1.3.2 2. Building MuPDFWrapper . . . 6
1.3.2.1 Windows (x86 and x64) . . . 7
1.3.2.2 Windows (arm64) . . . 7
1.3.2.3 macOS and Linux . . . 8
1.3.3 3. Creating the native assets MuPDFCore NuGet packages . . . 8
1.3.4 4. Creating the MuPDFCore NuGet package . . . 8
1.3.5 5. Running tests . . . 9
1.3.5.1 Windows . . . 10
1.3.5.2 macOS and Linux . . . 10
1.4 Note about MuPDFCore and .NET Framework . . . 10

2 Namespace Index 13
2.1 Package List . . . 13

3 Hierarchical Index 15
3.1 Class Hierarchy . . . 15

4 Class Index 17
4.1 Class List . . . 17

5 File Index 19
5.1 File List . . . 19

6 Namespace Documentation 21
6.1 Avalonia Namespace Reference . . . 21
6.2 Avalonia.Animation Namespace Reference . . . 21
6.3 MuPDFCore Namespace Reference . . . 21
6.3.1 Enumeration Type Documentation . . . 23
6.3.1.1 DocumentOutputFileTypes . . . 23
6.3.1.2 DocumentRestrictions . . . 23
6.3.1.3 EncryptionState . . . 23
6.3.1.4 ExitCodes . . . 23
6.3.1.5 InputFileTypes . . . 24
6.3.1.6 PasswordTypes . . . 24
6.3.1.7 PixelFormats . . . 24
6.3.1.8 RasterOutputFileTypes . . . 24

6.3.1.9 RestrictionState . . . 24
6.4 MuPDFCore.MuPDFRenderer Namespace Reference . . . 24

7 Class Documentation **25**

7.1 MuPDFCore.DisposableIntPtr Class Reference . . . 25
7.1.1 Detailed Description . . . 25
7.1.2 Constructor & Destructor Documentation . . . 26
7.1.2.1 DisposableIntPtr() [1/2] . . . 26
7.1.2.2 DisposableIntPtr() [2/2] . . . 26
7.1.3 Member Function Documentation . . . 26
7.1.3.1 Dispose() . . . 26
7.2 MuPDFCore.DocumentLockedException Class Reference . . . 27
7.2.1 Detailed Description . . . 27
7.3 MuPDFCore.MessageEventArgs Class Reference . . . 27
7.3.1 Detailed Description . . . 28
7.3.2 Constructor & Destructor Documentation . . . 28
7.3.2.1 MessageEventArgs() . . . 28
7.3.3 Property Documentation . . . 28
7.3.3.1 Message . . . 28
7.4 MuPDFCore.MuPDF Class Reference . . . 29
7.4.1 Detailed Description . . . 29
7.4.2 Member Function Documentation . . . 29
7.4.2.1 RedirectOutput() . . . 29
7.4.2.2 ResetOutput() . . . 30
7.4.3 Event Documentation . . . 30
7.4.3.1 StandardErrorMessage . . . 30
7.4.3.2 StandardOutputMessage . . . 30
7.5 MuPDFCore.MuPDFContext Class Reference . . . 30
7.5.1 Detailed Description . . . 31
7.5.2 Constructor & Destructor Documentation . . . 31
7.5.2.1 MuPDFContext() . . . 31
7.5.3 Member Function Documentation . . . 32
7.5.3.1 ClearStore() . . . 32
7.5.3.2 Dispose() . . . 32
7.5.3.3 ShrinkStore() . . . 32
7.5.4 Property Documentation . . . 32
7.5.4.1 AntiAliasing . . . 32
7.5.4.2 GraphicsAntiAliasing . . . 33
7.5.4.3 StoreMaxSize . . . 33
7.5.4.4 StoreSize . . . 33
7.5.4.5 TextAntiAliasing . . . 33
7.6 MuPDFCore.MuPDFDocument Class Reference . . . 34

7.6.1 Detailed Description . . . 36
7.6.2 Constructor & Destructor Documentation . . . 37
7.6.2.1 MuPDFDocument() [1/5] . . . 37
7.6.2.2 MuPDFDocument() [2/5] . . . 37
7.6.2.3 MuPDFDocument() [3/5] . . . 38
7.6.2.4 MuPDFDocument() [4/5] . . . 38
7.6.2.5 MuPDFDocument() [5/5] . . . 38
7.6.3 Member Function Documentation . . . 39
7.6.3.1 ClearCache() . . . 39
7.6.3.2 CreateDocument() [1/2] . . . 39
7.6.3.3 CreateDocument() [2/2] . . . 39
7.6.3.4 Dispose() . . . 40
7.6.3.5 ExtractText() [1/2] . . . 40
7.6.3.6 ExtractText() [2/2] . . . 41
7.6.3.7 ExtractTextAsync() . . . 41
7.6.3.8 GetMultiThreadedRenderer() . . . 42
7.6.3.9 GetRenderedSize() [1/2] . . . 42
7.6.3.10 GetRenderedSize() [2/2] . . . 43
7.6.3.11 GetStructuredTextPage() [1/2] . . . 43
7.6.3.12 GetStructuredTextPage() [2/2] . . . 44
7.6.3.13 GetStructuredTextPageAsync() . . . 44
7.6.3.14 Layout() . . . 45
7.6.3.15 LayoutSinglePage() . . . 45
7.6.3.16 Render() [1/6] . . . 46
7.6.3.17 Render() [2/6] . . . 46
7.6.3.18 Render() [3/6] . . . 47
7.6.3.19 Render() [4/6] . . . 47
7.6.3.20 Render() [5/6] . . . 48
7.6.3.21 Render() [6/6] . . . 48
7.6.3.22 SaveImage() [1/2] . . . 49
7.6.3.23 SaveImage() [2/2] . . . 50
7.6.3.24 SaveImageAsJPEG() [1/2] . . . 50
7.6.3.25 SaveImageAsJPEG() [2/2] . . . 51
7.6.3.26 TryUnlock() [1/2] . . . 51
7.6.3.27 TryUnlock() [2/2] . . . 52
7.6.3.28 WriteImage() [1/2] . . . 52
7.6.3.29 WriteImage() [2/2] . . . 53
7.6.3.30 WriteImageAsJPEG() [1/2] . . . 53
7.6.3.31 WriteImageAsJPEG() [2/2] . . . 54
7.6.4 Property Documentation . . . 54
7.6.4.1 ClipToPageBounds . . . 54
7.6.4.2 EncryptionState . . . 54

7.6.4.3 Pages . . . 55
7.6.4.4 Restrictions . . . 55
7.6.4.5 RestrictionState . . . 55
7.7 MuPDFCore.MuPDFException Class Reference . . . 55
7.7.1 Detailed Description . . . 56
7.7.2 Member Data Documentation . . . 56
7.7.2.1 ErrorCode . . . 56
7.8 MuPDFCore.MuPDFImageStructuredTextBlock Class Reference . . . 56
7.8.1 Detailed Description . . . 57
7.8.2 Member Function Documentation . . . 57
7.8.2.1 GetEnumerator() . . . 57
7.8.3 Property Documentation . . . 57
7.8.3.1 Count . . . 57
7.8.3.2 this[int index] . . . 58
7.8.3.3 Type . . . 58
7.9 MuPDFCore.MuPDFMultiThreadedPageRenderer Class Reference . . . 58
7.9.1 Detailed Description . . . 59
7.9.2 Member Function Documentation . . . 59
7.9.2.1 Abort() . . . 59
7.9.2.2 Dispose() . . . 59
7.9.2.3 GetProgress() . . . 59
7.9.2.4 GetSpanItem() . . . 59
7.9.2.5 Render() [1/2] . . . 60
7.9.2.6 Render() [2/2] . . . 60
7.9.3 Property Documentation . . . 61
7.9.3.1 ThreadCount . . . 61
7.10 MuPDFCore.MuPDFPage Class Reference . . . 61
7.10.1 Detailed Description . . . 62
7.10.2 Member Function Documentation . . . 62
7.10.2.1 Dispose() . . . 62
7.10.3 Property Documentation . . . 62
7.10.3.1 Bounds . . . 62
7.10.3.2 PageNumber . . . 63
7.11 MuPDFCore.MuPDFPageCollection Class Reference . . . 63
7.11.1 Detailed Description . . . 64
7.11.2 Member Function Documentation . . . 64
7.11.2.1 Dispose() . . . 64
7.11.2.2 GetEnumerator() . . . 64
7.11.3 Property Documentation . . . 64
7.11.3.1 Count . . . 64
7.11.3.2 Length . . . 64
7.11.3.3 this[int index] . . . 64

7.12 MuPDFCore.MuPDFStructuredTextAddress Struct Reference . . . 65
7.12.1 Detailed Description . . . 66
7.12.2 Constructor & Destructor Documentation . . . 66
7.12.2.1 MuPDFStructuredTextAddress() . . . 66
7.12.3 Member Function Documentation . . . 67
7.12.3.1 CompareTo() . . . 67
7.12.3.2 Equals() [1/2] . . . 67
7.12.3.3 Equals() [2/2] . . . 68
7.12.3.4 GetHashCode() . . . 68
7.12.3.5 Increment() . . . 68
7.12.3.6 operator"!=() . . . 68
7.12.3.7 operator<() . . . 69
7.12.3.8 operator<=() . . . 69
7.12.3.9 operator==() . . . 70
7.12.3.10 operator>() . . . 70
7.12.3.11 operator>=() . . . 70
7.12.4 Member Data Documentation . . . 71
7.12.4.1 BlockIndex . . . 71
7.12.4.2 CharacterIndex . . . 71
7.12.4.3 LineIndex . . . 71
7.13 MuPDFCore.MuPDFStructuredTextAddressSpan Class Reference . . . 72
7.13.1 Detailed Description . . . 72
7.13.2 Constructor & Destructor Documentation . . . 72
7.13.2.1 MuPDFStructuredTextAddressSpan() . . . 72
7.13.3 Member Data Documentation . . . 72
7.13.3.1 End . . . 73
7.13.3.2 Start . . . 73
7.14 MuPDFCore.MuPDFStructuredTextBlock Class Reference . . . 73
7.14.1 Detailed Description . . . 74
7.14.2 Member Enumeration Documentation . . . 74
7.14.2.1 Types . . . 74
7.14.3 Member Function Documentation . . . 74
7.14.3.1 GetEnumerator() . . . 74
7.14.4 Property Documentation . . . 75
7.14.4.1 BoundingBox . . . 75
7.14.4.2 Count . . . 75
7.14.4.3 this[int index] . . . 75
7.14.4.4 Type . . . 75
7.15 MuPDFCore.MuPDFStructuredTextCharacter Class Reference . . . 76
7.15.1 Detailed Description . . . 76
7.15.2 Member Function Documentation . . . 76
7.15.2.1 ToString() . . . 76

7.15.3 Property Documentation . . . 77
7.15.3.1 BoundingQuad . . . 77
7.15.3.2 Character . . . 77
7.15.3.3 CodePoint . . . 77
7.15.3.4 Color . . . 77
7.15.3.5 Origin . . . 77
7.15.3.6 Size . . . 78
7.16 MuPDFCore.MuPDFStructuredTextLine Class Reference . . . 78
7.16.1 Detailed Description . . . 79
7.16.2 Member Enumeration Documentation . . . 79
7.16.2.1 WritingModes . . . 79
7.16.3 Member Function Documentation . . . 79
7.16.3.1 GetEnumerator() . . . 79
7.16.3.2 ToString() . . . 80
7.16.4 Property Documentation . . . 80
7.16.4.1 BoundingBox . . . 80
7.16.4.2 Characters . . . 80
7.16.4.3 Count . . . 80
7.16.4.4 Direction . . . 81
7.16.4.5 Text . . . 81
7.16.4.6 this[int index] . . . 81
7.16.4.7 WritingMode . . . 81
7.17 MuPDFCore.MuPDFStructuredTextPage Class Reference . . . 82
7.17.1 Detailed Description . . . 83
7.17.2 Member Function Documentation . . . 83
7.17.2.1 GetClosestHitAddress() . . . 83
7.17.2.2 GetEnumerator() . . . 83
7.17.2.3 GetHighlightQuads() . . . 83
7.17.2.4 GetHitAddress() . . . 84
7.17.2.5 GetText() . . . 84
7.17.2.6 Search() . . . 85
7.17.3 Property Documentation . . . 85
7.17.3.1 Count . . . 85
7.17.3.2 StructuredTextBlocks . . . 85
7.17.3.3 this[int index] . . . 85
7.17.3.4 this[MuPDFStructuredTextAddress address] . . . 86
7.18 MuPDFCore.MuPDFTextStructuredTextBlock Class Reference . . . 86
7.18.1 Detailed Description . . . 87
7.18.2 Member Function Documentation . . . 87
7.18.2.1 GetEnumerator() . . . 88
7.18.2.2 ToString() . . . 88
7.18.3 Property Documentation . . . 88

7.18.3.1 Count . . . 88
7.18.3.2 Lines . . . 88
7.18.3.3 this[int index] . . . 89
7.18.3.4 Type . . . 89
7.19 MuPDFCore.OCRProgressInfo Class Reference . . . 89
7.19.1 Detailed Description . . . 89
7.19.2 Property Documentation . . . 89
7.19.2.1 Progress . . . 89
7.20 MuPDFCore.MuPDFRenderer.PDFRenderer Class Reference . . . 90
7.20.1 Detailed Description . . . 93
7.20.2 Member Enumeration Documentation . . . 93
7.20.2.1 PointerEventHandlers . . . 93
7.20.3 Constructor & Destructor Documentation . . . 93
7.20.3.1 PDFRenderer() . . . 94
7.20.4 Member Function Documentation . . . 94
7.20.4.1 Contain() . . . 94
7.20.4.2 Cover() . . . 94
7.20.4.3 GetProgress() . . . 94
7.20.4.4 GetSelectedText() . . . 95
7.20.4.5 Initialize() [1/4] . . . 95
7.20.4.6 Initialize() [2/4] . . . 96
7.20.4.7 Initialize() [3/4] . . . 96
7.20.4.8 Initialize() [4/4] . . . 97
7.20.4.9 InitializeAsync() [1/4] . . . 97
7.20.4.10 InitializeAsync() [2/4] . . . 98
7.20.4.11 InitializeAsync() [3/4] . . . 99
7.20.4.12 InitializeAsync() [4/4] . . . 100
7.20.4.13 ReleaseResources() . . . 100
7.20.4.14 Render() . . . 101
7.20.4.15 Search() . . . 101
7.20.4.16 SelectAll() . . . 101
7.20.4.17 SetDisplayAreaNow() . . . 101
7.20.4.18 ZoomStep() . . . 102
7.20.5 Member Data Documentation . . . 102
7.20.5.1 BackgroundProperty . . . 102
7.20.5.2 DisplayAreaProperty . . . 102
7.20.5.3 HighlightBrushProperty . . . 103
7.20.5.4 HighlightedRegionsProperty . . . 103
7.20.5.5 IsViewerInitializedProperty . . . 103
7.20.5.6 PageBackgroundProperty . . . 103
7.20.5.7 PageNumberProperty . . . 104
7.20.5.8 PageSizeProperty . . . 104

7.20.5.9 PointerEventHandlerTypeProperty . . . 104
7.20.5.10 RenderThreadCountProperty . . . 104
7.20.5.11 SelectionBrushProperty . . . 105
7.20.5.12 SelectionProperty . . . 105
7.20.5.13 ZoomEnabledProperty . . . 105
7.20.5.14 ZoomIncrementProperty . . . 105
7.20.5.15 ZoomProperty . . . 106
7.20.6 Property Documentation . . . 106
7.20.6.1 Background . . . 106
7.20.6.2 DisplayArea . . . 106
7.20.6.3 HighlightBrush . . . 106
7.20.6.4 HighlightedRegions . . . 107
7.20.6.5 IsViewerInitialized . . . 107
7.20.6.6 PageBackground . . . 107
7.20.6.7 PageNumber . . . 107
7.20.6.8 PageSize . . . 107
7.20.6.9 PointerEventHandlersType . . . 108
7.20.6.10 RenderThreadCount . . . 108
7.20.6.11 Selection . . . 108
7.20.6.12 SelectionBrush . . . 108
7.20.6.13 Zoom . . . 108
7.20.6.14 ZoomEnabled . . . 109
7.20.6.15 ZoomIncrement . . . 109
7.21 MuPDFCore.PointF Struct Reference . . . 109
7.21.1 Detailed Description . . . 109
7.21.2 Constructor & Destructor Documentation . . . 109
7.21.2.1 PointF() . . . 109
7.21.3 Member Data Documentation . . . 110
7.21.3.1 X . . . 110
7.21.3.2 Y . . . 110
7.22 MuPDFCore.Quad Struct Reference . . . 110
7.22.1 Detailed Description . . . 111
7.22.2 Constructor & Destructor Documentation . . . 111
7.22.2.1 Quad() . . . 111
7.22.3 Member Function Documentation . . . 111
7.22.3.1 Contains() . . . 112
7.22.4 Member Data Documentation . . . 112
7.22.4.1 LowerLeft . . . 112
7.22.4.2 LowerRight . . . 112
7.22.4.3 UpperLeft . . . 112
7.22.4.4 UpperRight . . . 113
7.23 MuPDFCore.Rectangle Struct Reference . . . 113

7.23.1 Detailed Description . . . 114
7.23.2 Constructor & Destructor Documentation . . . 114
7.23.2.1 Rectangle() [1/2] . . . 114
7.23.2.2 Rectangle() [2/2] . . . 114
7.23.3 Member Function Documentation . . . 115
7.23.3.1 Contains() [1/2] . . . 115
7.23.3.2 Contains() [2/2] . . . 115
7.23.3.3 Intersect() . . . 116
7.23.3.4 Round() [1/2] . . . 116
7.23.3.5 Round() [2/2] . . . 116
7.23.3.6 Split() . . . 117
7.23.3.7 ToQuad() . . . 117
7.23.4 Member Data Documentation . . . 117
7.23.4.1 X0 . . . 118
7.23.4.2 X1 . . . 118
7.23.4.3 Y0 . . . 118
7.23.4.4 Y1 . . . 118
7.23.5 Property Documentation . . . 118
7.23.5.1 Height . . . 118
7.23.5.2 Width . . . 119
7.24 Avalonia.Animation.RectTransition Class Reference . . . 119
7.24.1 Detailed Description . . . 119
7.25 MuPDFCore.RenderProgress Class Reference . . . 119
7.25.1 Detailed Description . . . 120
7.25.2 Property Documentation . . . 120
7.25.2.1 ThreadRenderProgresses . . . 120
7.26 MuPDFCore.RoundedRectangle Struct Reference . . . 120
7.26.1 Detailed Description . . . 121
7.26.2 Constructor & Destructor Documentation . . . 121
7.26.2.1 RoundedRectangle() . . . 121
7.26.3 Member Function Documentation . . . 122
7.26.3.1 Split() . . . 122
7.26.4 Member Data Documentation . . . 122
7.26.4.1 X0 . . . 122
7.26.4.2 X1 . . . 122
7.26.4.3 Y0 . . . 123
7.26.4.4 Y1 . . . 123
7.26.5 Property Documentation . . . 123
7.26.5.1 Height . . . 123
7.26.5.2 Width . . . 123
7.27 MuPDFCore.RoundedSize Struct Reference . . . 123
7.27.1 Detailed Description . . . 124

7.27.2 Constructor & Destructor Documentation . . . 124
7.27.2.1 RoundedSize() . . . 124
7.27.3 Member Function Documentation . . . 124
7.27.3.1 Split() . . . 125
7.27.4 Member Data Documentation . . . 125
7.27.4.1 Height . . . 125
7.27.4.2 Width . . . 125
7.28 MuPDFCore.Size Struct Reference . . . 125
7.28.1 Detailed Description . . . 126
7.28.2 Constructor & Destructor Documentation . . . 126
7.28.2.1 Size() [1/2] . . . 126
7.28.2.2 Size() [2/2] . . . 126
7.28.3 Member Function Documentation . . . 127
7.28.3.1 Split() . . . 127
7.28.4 Member Data Documentation . . . 127
7.28.4.1 Height . . . 127
7.28.4.2 Width . . . 128
7.29 MuPDFCore.TesseractLanguage Class Reference . . . 128
7.29.1 Detailed Description . . . 129
7.29.2 Member Enumeration Documentation . . . 129
7.29.2.1 Best . . . 129
7.29.2.2 BestScripts . . . 129
7.29.2.3 Fast . . . 129
7.29.2.4 FastScripts . . . 130
7.29.3 Constructor & Destructor Documentation . . . 130
7.29.3.1 TesseractLanguage() [1/6] . . . 130
7.29.3.2 TesseractLanguage() [2/6] . . . 130
7.29.3.3 TesseractLanguage() [3/6] . . . 131
7.29.3.4 TesseractLanguage() [4/6] . . . 131
7.29.3.5 TesseractLanguage() [5/6] . . . 131
7.29.3.6 TesseractLanguage() [6/6] . . . 132
7.29.4 Property Documentation . . . 132
7.29.4.1 Language . . . 132
7.29.4.2 Prefix . . . 133
7.30 MuPDFCore.RenderProgress.ThreadRenderProgress Struct Reference . . . 133
7.30.1 Detailed Description . . . 133
7.30.2 Member Data Documentation . . . 133
7.30.2.1 MaxProgress . . . 133
7.30.2.2 Progress . . . 133

8 File Documentation **135**

8.1 PDFRenderer.cs . . . 135

8.2 PDFRenderer.Properties.cs . . . 155
8.3 RectTransition.cs . . . 159
8.4 MuPDF.cs . . . 160
8.5 MuPDFContext.cs . . . 179
8.6 MuPDFDisplayList.cs . . . 181
8.7 MuPDFDocument.cs . . . 183
8.8 MuPDFMultiThreadedPageRenderer.cs . . . 206
8.9 MuPDFPage.cs . . . 214
8.10 MuPDFStructuredTextPage.cs . . . 216
8.11 Rectangles.cs . . . 233
8.12 TesseractLanguage.cs . . . 241
8.13 Utils.cs . . . 261

Index **265**

Chapter 1

MuPDFCore: Multiplatform .NET bindings for MuPDF

MuPDFCore is a set of multiplatform .NET bindings for `MuPDF`. It can render PDF, XPS, EPUB and other formats to raster images returned either as raw bytes, or as image files in multiple formats (including PNG, JPEG, and PSD). It also supports multithreading.

It also includes **MuPDFCore.MuPDFRenderer**, an Avalonia control to display documents compatible with MuPDFCore in Avalonia windows (with multithreaded rendering).

The library is released under the `AGPLv3` licence.

1.1 Getting started

The MuPDFCore library targets .NET Standard 2.0, thus it can be used in projects that target .NET Standard 2.0+, .NET Core 2.0+, .NET 5.0+, .NET Framework 4.6.1 (note) and possibly others. MuPDFCore includes a pre-compiled native library, which currently supports the following platforms:

- Windows x86 (32 bit) `win-x86`
- Windows x64 (64 bit) `win-x64`
- Windows arm64 (ARM 64 bit) `win-arm64`
- Linux x64 (64 bit)
  - glibc-based `linux-x64`
  - musl-based `linux-musl-x64`
- Linux arm64/aarch64 (ARM 64 bit)
  - glibc-based `linux-arm64`
  - musl-based `linux-musl-arm64` (see [note](muslNote))
- macOS Intel x86_64 (64 bit) `osx-x64`
- macOS Apple silicon (ARM 64 bit) `osx-arm64`

To use the library in your project, you should install the MuPDFCore NuGet package and/or the MuPDFCore.PDFRenderer NuGet package. When you publish a program that uses MuPDFCore, the correct native library for the target architecture will automatically be copied to the build folder (but see the note for .NET Framework).

Note: you should make sure that end users on **Windows** install the Microsoft Visual C++ Redistributable for Visual Studio 2015, 2017, 2019 and 2022 for their platform, otherwise they will get an error message stating that `MuPDFWrapper.dll` could not be loaded because a module was not found.

Note for **musl-based Linux arm64**: I could not find a way to ensure that the linux-musl-arm64 native artifact overwrites the linux-arm64 (`glibc`) artifact. As a result, when you publish a project that uses MuPDFCore targeting linux-musl-arm64, you will find *two* native assets in the build directory (`MuPDFWrapper.so`, which is the `musl` artifact, and `libMuPDFWrapper.so`, which is the `glibc` artifact). Everything will work fine out of the box (because the name of the `musl` artifact has higher priority), but you may want to delete `libMuPDFWrapper.so` in order to reduce size. You can use e.g. a post-build target to do this.

1.2 Usage

1.2.1 Documentation

You can find detailed descriptions of how to use MuPDFCore and some code examples in the MuPDFCore wiki. Interactive documentation for the library API can be accessed from the documentation website. A PDF reference manual is also available.

1.2.2 Minimal working example

The following example shows the bare minimum code necessary to render a page of a PDF document to a PNG image using MuPDFCore:

```
//Initialise the MuPDF context.  This is needed to open or create documents.
using MuPDFContext ctx = new MuPDFContext();
//Open a PDF document
using MuPDFDocument document = new MuPDFDocument(ctx, "path/to/document.pdf");
//Page index (page 0 is the first page of the document)
int pageIndex = 0;
//Zoom level, converting document units into pixels.  For a PDF document, a 1x zoom level corresponds to a
//72dpi resolution.
double zoomLevel = 1;
//Save the first page as a PNG image with transparency, at a 1x zoom level (1pt = 1px).
document.SaveImage(pageIndex, zoomLevel, PixelFormats.RGBA, "path/to/output.png",
      RasterOutputFileTypes.PNG);
```

Look at the wiki for more information.

1.2.3 Examples

The Demo folder in the repository contains some examples of how the library can be used to extract pages from a PDF or XPS document, render them to a raster image, or combine them in a new document

The PDFViewerDemo folder contains a complete (though minimal) example of a PDF viewer program built around the `MuPDFCore.MuPDFRenderer.PDFRenderer` control.

Note that these examples intentionally avoid any error handling code: in a production setting, you should typically make sure that calls to MuPDFCore library functions are within a `try...catch` block to handle any resulting `MuPDFException`s.

1.3 Building from source

Building the MuPDFCore library from source requires the following steps:

1. Building the `libmupdf` native library
2. Building the `MuPDFWrapper` native library
3. Creating the `MuPDFCore.NativeAssets.xxx-yyy` native assets NuGet packages
4. Creating the `MuPDFCore` library NuGet package

Starting from MuPDFCore 1.8.0, the native assets are split into their own NuGet packages, on which the main MuPDFCore package depends. Aside from reducing the size of individual packages, this means that if you are making changes that do not affect the native assets, you can skip steps 1-3 and go straight to step 4.

Steps 1 and 2 need to be performed on all of Windows, macOS and Linux, and on the various possible architectures (x86, x64 and arm64 for Windows, x64/Intel and arm64/Apple for macOS, and x64 and arm64 for Linux, both glibc and musl - no cross-compiling)! Otherwise, some native assets will be missing and it will not be possible to build the NuGet packages in step 3.

1.3.1 1. Building libmupdf

You can download the open-source (GNU AGPL) MuPDF source code from `here`. You will need to uncompress the source file and compile the library on Windows, macOS and Linux. You need the following files:

- From Windows (x86, x64, arm64):
  - libmupdf.lib
- From macOS (Intel - x64, Apple silicon - arm64):
  - libmupdf.a
  - libmupdf-third.a
- From Linux (x64, arm64):
  - libmupdf.a
  - libmupdf-third.a

Note that the files from macOS and Linux are different, despite sharing the same name.

For convenience, these compiled files for MuPDF 1.23.6 are included in the `native/MuPDFWrapper/lib folder` of this repository.

Tips for compiling MuPDF 1.23.6

- On all platforms:
  - You do not need to follow the instructions in `thirdparty/tesseract.txt`, as in this version the *leptonica* and *tesseract* libraries are already included in the source archive.

  - Delete or comment line 316 in `source/fitz/output.c` (the `fz_throw` invocation within the `buffer_seek` method - this should leave the `buffer_seek` method empty). This line throws an exception when a seek operation on a buffer is attempted. The problem is that this makes it impossible to render a document as a PSD image in memory, because the `fz_write_pixmap_as_psd` method performs a few seek operations. By removing this line, we turn buffer seeks into no-ops; this doesn't seem to have catastrophic side-effects and the PSD documents produced in this way appear to be fine.

- On Windows (x64):
  - Open the `platform/win32/mupdf.sln` solution in Visual Studio. You should get a prompt to retarget your projects. Accept the default settings (latest Windows SDK and v143 of the tools).
  - Select the `ReleaseTesseract` configuration and `x64` architecture. Select every project in the solution except `javaviewer` and `javaviewerlib` and right-click to open the project properties. Go to `C/C++ > Code Generation` and set the `Runtime Library` to `Multi-threaded DLL (/MD)`. Save everything (`CTRL+SHIFT+S`) and close Visual Studio.
  - Now, open the `x64 Native Tools Command Prompt for VS`, move to the folder with the solution file, and build it using `msbuild mupdf.sln`
  - Then, build again using `msbuild mupdf.sln /p:Configuration=Release`. Ignore the compilation errors.
  - Finally, build again using `msbuild mupdf.sln /p:Configuration=ReleaseTesseract`.
  - This may still show some errors, but should produce the `libmupdf.lib` file that is required in the `x64/ReleaseTesseract` folder (the file should be ~510MB in size).

- On Windows (x86):
  - You will have to use Visual Studio 2019, as Visual Studio 2022 is not supported on x86 platforms.
  - Open the `platform/win32/mupdf.sln` solution in Visual Studio and select the `Release↩ Tesseract` configuration and `Win32` architecture. Select every project in the solution except `javaviewer` and `javaviewerlib` and right-click to open the project properties. Go to `C/C++ > Code Generation` and set the `Runtime Library` to `Multi-threaded DLL (/MD)`. Save everything (`CTRL+SHIFT+S`) and close Visual Studio.
  - Now, open the `x86 Native Tools Command Prompt for VS`, move to the folder with the solution file, and build it using `msbuild mupdf.sln /p:Platform=Win32`
  - Then, build again using `msbuild mupdf.sln /p:Configuration=Release /p↩ :Platform=Win32`. Ignore the compilation errors.
  - Finally, build again using `msbuild mupdf.sln /p:Configuration=ReleaseTesseract /p:Platform=Win32`.
  - This may still show some errors, but should produce the `libmupdf.lib` file that is required in the `ReleaseTesseract` folder (the file should be ~420MB in size).

- On Windows (arm64)

  This is going to be a bit more complicated, because it appears that MuPDF is not meant to be built on ARM. These instructions will assume that you are building MuPDF on an ARM machine.

  First of all, make sure that you have installed Visual Studio 2022 and have selected the C++ ARM64 build tools component of the "Desktop development with C++" workload.

  - Download and extract the MuPDF source code and follow the instructions for all platforms above.
  - Add `|| defined(_M_ARM64)` at the end of line 16 in `scripts/tesseract/endianness.h`.
  - Open the file `thirdparty/openjpeg/src/lib/openjp2/ht_dec.c` and add the following after line 57 (`source`): ```C unsigned int __popcnt(unsigned int x) { unsigned int c = 0; for (; x; ++c) { x &= x - 1; } return c; } ``
  - `Now we need to edit a few files in the`thirdparty/tesseract/src/arch`folder.`
    * `Comment or delete lines 149-177 (inclusive) insimddetect.cpp.`
      `You should now have an empty block between# elif defined(_WIN32)and#else.`
      `Also comment or delete lines 198-220 (inclusive) and 237-260 (inclusive).`

    * Comment or delete lines 20-22 (inclusive) indotproductsse.cpp. Replace the whole body of theDotProductSSEmethod (lines 30-76) withreturn DotProductNative(u, v, n);.
    * Comment or delete lines 20-21 (inclusive) indotproductavx.cpp. Replace the whole body of theDotProductAVXmethod (lines 29-54) withreturn DotProductNative(u, v, n);.
    * Comment or delete lines 20-21 (inclusive) indotproductfma.cpp. Replace the whole body of theDotProductFMAmethod (lines 29-52) withreturn DotProductNative(u, v, n);.
    * Delete the contents ofthirdparty/tesseract/src/arch/intsimdmatrixavx2.↩ cppandthirdparty/tesseract/src/arch/intsimdmatrixsse.cpp(do not delete the files, just their contents).
    * Comment or delete lines 120-121 (inclusive) inintsimdmatrix.h`

  - Open the platform/win32/mupdf.sln solution in Visual Studio. You should get a prompt to retarget your projects. Accept the default settings (latest Windows SDK and v143 of the tools).
  - In Visual Studio, click on the "Configuration Manager" item from the "Build" menu. In the new window, click on the drop down menu for the "Active solution platform" and select <New...>. In this new dialog, select the ARM64 platform and choose to copy the settings from x64. Leave the Create new project platforms option enabled and click on OK (this may take some time).
  - Close the Configuration Manager and select the ReleaseTesseract configuration and ARM64 architecture. Select every project in the solution except javaviewer and javaviewerlib and right-click to open the project properties. Go to C/C++ > Code Generation and set the Runtime Library to Multi-threaded DLL (/MD).
  - Open the properties for the libpkcs7 project, go to C/C++ > Preprocessor and remove HAVE_LIBCRYPTO from the Preprocessor Definitions. Then go to Librarian > General and remove libcrypto.lib from the Additional Dependencies.
  - Save everything (CTRL+SHIFT+S) and close Visual Studio.
  - Create a new folder platform/win32/Release. Now, the problem is that the bin2coff script included with MuPDF cannot create obj files for ARM64 (only for x86 and x64). Since I could not find a version that can do this, I translated the source code of bin2coff to C# and added this option myself. You can download an ARM64 bin2coff.↩ exe from here; place it in the Release folder that you have just created.
  - Open the Developer Command Prompt for VS, move to the folder with the solution file (platform/win32), and build it using msbuild mupdf.↩ sln /p:Configuration=ReleaseTesseract. Some compilation errors may occur towards the end, but they should not matter.
  - After a while, this should produce libmupdf.lib in the ARM64/Release↩ Tesseract folder (the file should be ~506MB in size).

- On Linux (x64, for both glibc- and musl- based distros):

  - Edit the Makefile, adding the -fPIC compiler option at the end of line 24 (which specifies the CFLAGS).
  - Comment line 218 in include/mupdf/fitz/config.h (for some reason, this seems to disable OCR even when using USE_TESSERACT=yes to build).
  - Make sure that you are using a recent enough version of GCC (version 7.3.1 seems to be enough).

  - Compile by running USE_TESSERACT=yes make HAVE_X11=no HAVE_GLUT=no (this builds just the command-line libraries and tools, and enables OCR through the included Tesseract library).

- On Linux (arm64, for both glibc- and musl- based distros):
  - Edit the Makefile, adding the -fPIC compiler option at the end of line 24 (which specifies the CFLAGS).
  - Delete or comment line 218 in thirdparty/tesseract/src/arch/simddetect.↩ cpp.
  - Make sure that you are using a recent enough version of GCC (version 7.3.1 seems to be enough).
  - Compile by running USE_TESSERACT=yes make HAVE_X11=no HAVE_GLUT=no (this builds just the command-line libraries and tools, and enables OCR through the included Tesseract library).

- On macOS (Intel - x64):
  - Edit the Makefile, adding the -fPIC compiler option at the end of line 24 (which specifies the CFLAGS). Also add the -std=c++11 option at the end of line 58 (which specifies the CXX_CMD).
  - Compile by running USE_TESSERACT=yes make (this enables OCR through the included Tesseract library).

- On macOS (Apple silicon - arm64)
  - Edit the Makefile, adding the -fPIC compiler options at the end of line 24 (which specifies the CFLAGS). Also add the -std=c++11 option at the end of line 58 (which specifies the CXX_CMD).
  - Delete or comment line 218 in thirdparty/tesseract/src/arch/simddetect.↩ cpp.
  - Compile by running USE_TESSERACT=yes make (this enables OCR through the included Tesseract library).

1.3.2 2. Building MuPDFWrapper

Once you have the required static library files, you should download the MuPDFCore source code (just clone this repository) and place the library files in the appropriate subdirectories in the native/MuPDFWrapper/lib/ folder (for Linux x64, copy the library built agains glibc to the linux-x64 folder, and the library built against musl to the linux-musl-x64 folder, and do the same for Linux arm64).

To compile MuPDFWrapper you will need CMake (version 3.8 or higher) and (on Windows) Ninja.

On Windows, the easiest way to get all the required tools is probably to install Visual Studio. By selecting the "Desktop development with C++" workload you should get everything you need.

On macOS, you will need to install at least the Command-Line Tools for Xcode (if necessary, you should be prompted to do this while you perform the following steps) and CMake.

Once you have everything at the ready, you will have to build MuPDFWrapper on the seven platforms.

Build instructions

1.3.2.1 Windows (x86 and x64)

1.

Assuming you have installed Visual Studio, you should open the "x64 Native Tools Command Prompt for VS" or the "x86 Native Tools Command Prompt for VS" (you should be able to find these in the Start menu). Take care to open the version corresponding to the architecture you are building for, otherwise you will not be able to compile the library. A normal command prompt will not work, either.

Note 1: you **must** build the library on two separate systems, one running a 32-bit version of Windows and the other running a 64-bit version. If you try to build the x86 library on an x64 system, the system will probably build a 64-bit library and place it in the 32-bit output folder, which will just make things very confusing.

Note 2 for Windows x86: for some reason, Visual Studio might install the 64-bit version of CMake and Ninja, even though you are on a 32-bit machine. If this happens, you will have to manually install the 32-bit CMake and compile a 32-bit version of Ninja. You will notice if this is an issue because the 64-bit programs will refuse to run.

1. CD to the directory where you have downloaded the MuPDFCore source code.
2. CD into the native directory.
3. Type build. This will start the build.cmd batch script that will delete any previous build and compile the library.

After this finishes, you should find a file named MuPDFWrapper.dll in the native/out/build/win-x64/MuPDFWrapper/ directory or in the native/out/build/win-x86/↩ MuPDFWrapper/ directory. Leave it there.

1.3.2.2 Windows (arm64)

1. Locate the batch file that sets up the developer command prompt environment. You can do this by finding the "Developer Command Prompt for VS" link in the start menu, then clicking on Open file location, opening the properties of the link and looking at the Target. This could be e.g. C:\Program Files\Microsoft Visual Studio\2022\Preview\Common7\Tools\Vs↩ DevCmd.bat.
2. Open a normal command prompt and invoke the batch script with the -arch=arm64 -host_arch=x86 arguments (add quotes if there are spaces in the path to the batch script), e.g.:
   ```
   "C:\Program Files\Microsoft Visual Studio\2022\Preview\Common7\Tools\VsDevCmd.bat" -arch=arm64 -host_arch=x86
   ```
3. CD to the directory where you have downloaded the MuPDFCore source code.
4. CD into the native directory.
5. Type build. This will start the build.cmd batch script that will delete any previous build and compile the library.

After this finishes, you should find a file named MuPDFWrapper.dll in the native/out/build/win-arm64/MuPDFWrapper/ directory. Leave it there.

1.3.2.3 macOS and Linux

1. Assuming you have everything ready, open a terminal in the folder where you have downloaded the MuPDFCore source code.
2. cd into the native directory.
3. Type chmod +x build.sh.
4. Type ./build.sh. This will delete any previous build and compile the library.

After this finishes, you should find a file named libMuPDFWrapper.dylib in the native/out/build/mac-x64/MuPDFWrapper/ directory (on macOS running on an Intel x64 processor) or in the native/out/build/mac-arm64/MuPDFWrapper/ directory (on macOS running on an Apple silicon arm64 processor), and a file named libMuPDFWrapper.so in the native/out/build/linux-XXX/MuPDFWrapper/ directory (on Linux - where XXX can be x64, arm64, musl-x64, or musl-arm64). Leave it there.

1.3.3 3. Creating the native assets MuPDFCore NuGet packages

Once you have the MuPDFWrapper.dll (3x), libMuPDFWrapper.dylib (2x) and libMuPDFWrapper.so (4x) files, make sure they are in the correct folders (native/out/build/xxx-yyy/MuPDFWrapper/), **all on the same machine.**

To create the native assets NuGet packages, you will need the .NET Core 2.0 SDK or higher for your platform. Once you have installed it and have everything ready, open a terminal in the folder where you have downloaded the MuPDFCore source code and type:

```
BuildNativeAssets
```

This will create the NuGet packages in the MuPDFCore.NativeAssets/NuGet↩ Packages folder. Once the script finishes, this folder should contain 9 files. Make sure you add this folder as a local NuGet source.

1.3.4 4. Creating the MuPDFCore NuGet package

If you have made updates to the native assets, make sure to use the appropriate version numbers in MuPDFCore/MuPDFCore.csproj. Then, to create the main MuPDFCore NuGet package, open a terminal in the folder where you have downloaded the MuPDFCore source code and type:

```
cd MuPDFCore
dotnet pack -c Release
```

This will create a NuGet package in MuPDFCore/bin/Release. You can install this package on your projects by adding a local NuGet source.

1.3.5 5. Running tests

To verify that everything is working correctly, you should build the MuPDFCore test suite and run it on all platforms. To build the test suite, you will need the .NET 6 SDK or higher. You will also need to have enabled the Windows Subsystem for Linux.

To build the test suite:

1. Make sure that you have changed the version of the MuPDFCore NuGet package so that it is higher than the latest version of MuPDFCore in the NuGet repository (you should use a pre-release suffix, e.g. 1.4.0-a1 to avoid future headaches with new versions of MuPDFCore). This is set in line 9 of the MuPDFCore/MuPDFCore.csproj file.
2. Add the MuPDFCore/bin/Release folder to your local NuGet repositories (you can do this e.g. in Visual Studio).
3. If you have not done so already, create the MuPDFCore NuGet package following step 3 above.
4. Update line 56 of the Tests/Tests.csproj project file so that it refers to the version of the MuPDFCore package you have just created.

These steps ensure that you are testing the right version of MuPDFCore (i.e. your freshly built copy) and not something else that may have been cached.

Now, open a Windows command line in the folder where you have downloaded the MuPDFCore source code, type BuildTests and press Enter. This will create a number of files in the Release\MuPDFCoreTests folder, where each file is an archive containing the tests for a certain platform and architecture:

- MuPDFCoreTests-linux-x64.tar.gz contains the tests for Linux environments using glibc on x64 processors.
- MuPDFCoreTests-linux-arm64.tar.gz contains the tests for Linux environments using glibc on arm64 processors.
- MuPDFCoreTests-linux-musl-x64.tar.gz contains the tests for Linux environments using musl on x64 processors.
- MuPDFCoreTests-linux-musl-arm64.tar.gz contains the tests for Linux environments using musl on arm64 processors.
- MuPDFCoreTests-mac-x64.tar.gz contains the tests for macOS environments on Intel processors.
- MuPDFCoreTests-mac-arm64.tar.gz contains the tests for macOS environments on Apple silicon processors.
- MuPDFCoreTests-win-x64.tar.gz contains the tests for Windows environments on x64 processors.
- MuPDFCoreTests-win-x86.tar.gz contains the tests for Windows environments on x86 processors.
- MuPDFCoreTests-win-arm64.tar.gz contains the tests for Windows environments on arm64 processors.

To run the tests, copy each archive to a machine running the corresponding operating system, and extract it (note: on Windows, the default zip file manager may struggle when extracting the text file with non-latin characters; you may need to manually extract this file). Then:

1.3.5.1 Windows

- Open a command prompt and CD into the folder where you have extracted the contents of the test archive.
- Enter the command MuPDFCoreTestHost (this will run the test program).

1.3.5.2 macOS and Linux

- Open a terminal and cd into the folder where you have extracted the contents of the test archive.
- Enter the command chmod +x MuPDFCoreTestHost (this will add the executable flag to the test program).
- Enter the command ./MuPDFCoreTestHost (this will run the test program).
- On macOS, depending on your security settings, you may get a message saying zsh: killed when you try to run the program. To address this, you need to sign the executable, e.g. by running codesign --timestamp --sign <certificate> MuPDFCoreTestHost, where <certificate> is the name of a code signing certificate in your keychain (e.g. Developer ID Application: John Smith). After this, you can try again to run the test program with ./MuPDFCoreTestHost.

The test suite will start; it will print the name of each test, followed by a green Succeeded or a red Failed depending on the test result. If everything went correctly, all tests should succeed.

When all the tests have been run, the program will print a summary showing how many tests have succeeded (if any) and how many have failed (if any). If any tests have failed, a list of these will be printed, and then they will be run again one at a time, waiting for a key press before running each test (this makes it easier to follow what is going on). If you wish to kill the test process early, you can do so with CTRL+C.

1.4 Note about MuPDFCore and .NET Framework

If you wish to use MuPDFCore 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/.NET core).

One way to obtain the appropriate library files is:

1. Manually download the appropriate native assets NuGet package from the table below. Note that AnyCPU builds on Windows need the win-x86 native asset.
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/native/ folder, where xxx is linux-x64, linux-arm64, linux-musl-x64, linux-musl-arm64, osx-x64, osx-arm64, win-x64, win-x86 or win-arm64, depending on the platform you are using.
5. The file you need to copy should be called MuPDFWrapper.dll on Windows, libMuPDFWrapper.so or MuPDFWrapper.so on Linux, and libMuPDFWrapper.↩ dylib on macOS.

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

OS

Platform

NuGet package </thead> <tbody>

Windows

x86

win-x86

x64

win-x64

arm64

win-arm64

Linux

x64

glibc

linux-x64

musl

linux-musl-x64

arm64

glibc

linux-arm64

musl

linux-musl-arm64

macOS

x64 (Intel)

osx-x64

arm64 (Apple Silicon)

osx-arm64 </tbody>

Chapter 2

Namespace Index

2.1 Package List

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

Avalonia 21
Avalonia.Animation 21
MuPDFCore 21
MuPDFCore.MuPDFRenderer 24

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Control
- MuPDFCore.MuPDFRenderer.PDFRenderer . . . 90
- MuPDFCore.MuPDFRenderer.PDFRenderer . . . 90

EventArgs
- MuPDFCore.MessageEventArgs . . . 27

Exception
- MuPDFCore.DocumentLockedException . . . 27
- MuPDFCore.MuPDFException . . . 55

IComparable
- MuPDFCore.MuPDFStructuredTextAddress . . . 65

IDisposable
- MuPDFCore.DisposableIntPtr . . . 25
- MuPDFCore.MuPDFContext . . . 30
- MuPDFCore.MuPDFDocument . . . 34
- MuPDFCore.MuPDFMultiThreadedPageRenderer . . . 58
- MuPDFCore.MuPDFPage . . . 61
- MuPDFCore.MuPDFPageCollection . . . 63

IEquatable
- MuPDFCore.MuPDFStructuredTextAddress . . . 65

InterpolatingTransitionBase
- Avalonia.Animation.RectTransition . . . 119

IReadOnlyList
- MuPDFCore.MuPDFPageCollection . . . 63
- MuPDFCore.MuPDFStructuredTextBlock . . . 73
  - MuPDFCore.MuPDFImageStructuredTextBlock . . . 56
  - MuPDFCore.MuPDFTextStructuredTextBlock . . . 86
- MuPDFCore.MuPDFStructuredTextLine . . . 78
- MuPDFCore.MuPDFStructuredTextPage . . . 82

MuPDFCore.MuPDF . . . 29
MuPDFCore.MuPDFStructuredTextAddressSpan . . . 72
MuPDFCore.MuPDFStructuredTextCharacter . . . 76
MuPDFCore.OCRProgressInfo . . . 89
MuPDFCore.PointF . . . 109
MuPDFCore.Quad . . . 110
MuPDFCore.Rectangle . . . 113

MuPDFCore.RenderProgress 119
MuPDFCore.RoundedRectangle 120
MuPDFCore.RoundedSize 123
MuPDFCore.Size 125
MuPDFCore.TesseractLanguage 128
MuPDFCore.RenderProgress.ThreadRenderProgress 133

Chapter 4

Class Index

4.1 Class List

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

MuPDFCore.DisposableIntPtr
: An IDisposable wrapper around an IntPtr that frees the allocated memory when it is disposed. 25

MuPDFCore.DocumentLockedException
: The exception that is thrown when an attempt is made to render an encrypted document without supplying the required password. 27

MuPDFCore.MessageEventArgs
: EventArgs for the MuPDF.StandardOutputMessage and MuPDF.StandardErrorMessage events. 27

MuPDFCore.MuPDF
: Contains static methods to perform setup operations. 29

MuPDFCore.MuPDFContext
: A wrapper around a MuPDF context object, which contains the exception stack and the resource cache store. 30

MuPDFCore.MuPDFDocument
: A wrapper over a MuPDF document object, which contains possibly multiple pages. 34

MuPDFCore.MuPDFException
: The exception that is thrown when a MuPDF operation fails. 55

MuPDFCore.MuPDFImageStructuredTextBlock
: Represents a block containing a single image. The block contains a single line with a single character. 56

MuPDFCore.MuPDFMultiThreadedPageRenderer
: A class that holds the necessary resources to render a page of a MuPDF document using multiple threads. 58

MuPDFCore.MuPDFPage
: A wrapper over a MuPDF page object, which contains information about the page's boundaries. 61

MuPDFCore.MuPDFPageCollection
: A lazy collection of MuPDFPages. Each page is loaded from the document as it is requested for the first time. 63

MuPDFCore.MuPDFStructuredTextAddress
: Represents the address of a particular character in a MuPDFStructuredTextPage, in terms of block index, line index and character index. 65

MuPDFCore.MuPDFStructuredTextAddressSpan
: Represents a range of characters in a MuPDFStructuredTextPage. 72

MuPDFCore.MuPDFStructuredTextBlock
: Represents a structured text block containing text or an image. 73

MuPDFCore.MuPDFStructuredTextCharacter
: Represents a single text character. 76

MuPDFCore.MuPDFStructuredTextLine
Represents a single line of text (i.e. characters that share a common baseline). 78
MuPDFCore.MuPDFStructuredTextPage
Represents a structured representation of the text contained in a page. 82
MuPDFCore.MuPDFTextStructuredTextBlock
Represents a block containing multiple lines of text (typically a paragraph). 86
MuPDFCore.OCRProgressInfo
Describes OCR progress. 89
MuPDFCore.MuPDFRenderer.PDFRenderer
A control to render PDF documents (and other formats), potentally using multiple threads. . . . 90
MuPDFCore.PointF
Represents a point. 109
MuPDFCore.Quad
Represents a quadrilater (not necessarily a rectangle). 110
MuPDFCore.Rectangle
Represents a rectangle. 113
Avalonia.Animation.RectTransition
Transition class that handles AvaloniaProperty with Rect types. 119
MuPDFCore.RenderProgress
Holds a summary of the progress of the current rendering operation. 119
MuPDFCore.RoundedRectangle
Represents a rectangle using only integer numbers. 120
MuPDFCore.RoundedSize
Represents the size of a rectangle using only integer numbers. 123
MuPDFCore.Size
Represents the size of a rectangle. 125
MuPDFCore.TesseractLanguage
Represents a language used by Tesseract OCR. 128
MuPDFCore.RenderProgress.ThreadRenderProgress
Holds the progress of a single thread. 133

Chapter 5

File Index

5.1 File List

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

MuPDFCore.MuPDFRenderer/PDFRenderer.cs . . . 135
MuPDFCore.MuPDFRenderer/PDFRenderer.Properties.cs . . . 155
MuPDFCore.MuPDFRenderer/RectTransition.cs . . . 159
MuPDFCore/MuPDF.cs . . . 160
MuPDFCore/MuPDFContext.cs . . . 179
MuPDFCore/MuPDFDisplayList.cs . . . 181
MuPDFCore/MuPDFDocument.cs . . . 183
MuPDFCore/MuPDFMultiThreadedPageRenderer.cs . . . 206
MuPDFCore/MuPDFPage.cs . . . 214
MuPDFCore/MuPDFStructuredTextPage.cs . . . 216
MuPDFCore/Rectangles.cs . . . 233
MuPDFCore/TesseractLanguage.cs . . . 241
MuPDFCore/Utils.cs . . . 261

Chapter 6

Namespace Documentation

6.1 Avalonia Namespace Reference

6.2 Avalonia.Animation Namespace Reference

Classes

- class RectTransition

  Transition class that handles AvaloniaProperty with Rect types.

6.3 MuPDFCore Namespace Reference

Classes

- class DisposableIntPtr

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

  The exception that is thrown when an attempt is made to render an encrypted document without supplying the required password.
- class MessageEventArgs

  EventArgs for the MuPDF.StandardOutputMessage and MuPDF.StandardErrorMessage events.
- class MuPDF

  Contains static methods to perform setup operations.
- class MuPDFContext

  A wrapper around a MuPDF context object, which contains the exception stack and the resource cache store.
- class MuPDFDocument

  A wrapper over a MuPDF document object, which contains possibly multiple pages.
- class MuPDFException

  The exception that is thrown when a MuPDF operation fails.
- class MuPDFImageStructuredTextBlock

  Represents a block containing a single image. The block contains a single line with a single character.
- class MuPDFMultiThreadedPageRenderer

  A class that holds the necessary resources to render a page of a MuPDF document using multiple threads.

- class MuPDFPage

  A wrapper over a MuPDF page object, which contains information about the page's boundaries.
- class MuPDFPageCollection

  A lazy collection of MuPDFPages. Each page is loaded from the document as it is requested for the first time.
- struct MuPDFStructuredTextAddress

  Represents the address of a particular character in a MuPDFStructuredTextPage, in terms of block index, line index and character index.
- class MuPDFStructuredTextAddressSpan

  Represents a range of characters in a MuPDFStructuredTextPage.
- class MuPDFStructuredTextBlock

  Represents a structured text block containing text or an image.
- class MuPDFStructuredTextCharacter

  Represents a single text character.
- class MuPDFStructuredTextLine

  Represents a single line of text (i.e. characters that share a common baseline).
- class MuPDFStructuredTextPage

  Represents a structured representation of the text contained in a page.
- class MuPDFTextStructuredTextBlock

  Represents a block containing multiple lines of text (typically a paragraph).
- class OCRProgressInfo

  Describes OCR progress.
- struct PointF

  Represents a point.
- struct Quad

  Represents a quadrilater (not necessarily a rectangle).
- struct Rectangle

  Represents a rectangle.
- class RenderProgress

  Holds a summary of the progress of the current rendering operation.
- struct RoundedRectangle

  Represents a rectangle using only integer numbers.
- struct RoundedSize

  Represents the size of a rectangle using only integer numbers.
- struct Size

  Represents the size of a rectangle.
- class TesseractLanguage

  Represents a language used by Tesseract OCR.

Enumerations

- enum ExitCodes

  Exit codes returned by native methods describing various errors that can occur.
- enum InputFileTypes

  File types supported in input by the library.
- enum RasterOutputFileTypes

  Raster image file types supported in output by the library.
- enum DocumentOutputFileTypes

  Document file types supported in output by the library.
- enum PixelFormats

  Pixel formats supported by the library.

- enum EncryptionState

  Possible document encryption states.
- enum RestrictionState

  Possible document restriction states.
- enum DocumentRestrictions

  Document restrictions.
- enum PasswordTypes

  Password types.

6.3.1 Enumeration Type Documentation

6.3.1.1 DocumentOutputFileTypes

`enum MuPDFCore.DocumentOutputFileTypes`

Document file types supported in output by the library.

Definition at line 229 of file MuPDF.cs.

6.3.1.2 DocumentRestrictions

`enum MuPDFCore.DocumentRestrictions`

Document restrictions.

Definition at line 318 of file MuPDF.cs.

6.3.1.3 EncryptionState

`enum MuPDFCore.EncryptionState`

Possible document encryption states.

Definition at line 276 of file MuPDF.cs.

6.3.1.4 ExitCodes

`enum MuPDFCore.ExitCodes`

Exit codes returned by native methods describing various errors that can occur.

Definition at line 31 of file MuPDF.cs.

6.3.1.5 InputFileTypes

enum MuPDFCore.InputFileTypes

File types supported in input by the library.

Definition at line 122 of file MuPDF.cs.

6.3.1.6 PasswordTypes

enum MuPDFCore.PasswordTypes

Password types.

Definition at line 349 of file MuPDF.cs.

6.3.1.7 PixelFormats

enum MuPDFCore.PixelFormats

Pixel formats supported by the library.

Definition at line 250 of file MuPDF.cs.

6.3.1.8 RasterOutputFileTypes

enum MuPDFCore.RasterOutputFileTypes

Raster image file types supported in output by the library.

Definition at line 198 of file MuPDF.cs.

6.3.1.9 RestrictionState

enum MuPDFCore.RestrictionState

Possible document restriction states.

Definition at line 297 of file MuPDF.cs.

6.4 MuPDFCore.MuPDFRenderer Namespace Reference

Classes

- class PDFRenderer

  A control to render PDF documents (and other formats), potentally using multiple threads.

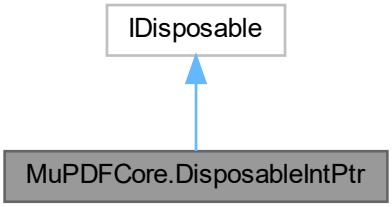
Chapter 7

Class Documentation

7.1 MuPDFCore.DisposableIntPtr Class Reference

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

Inheritance diagram for MuPDFCore.DisposableIntPtr:



Public Member Functions

- DisposableIntPtr (IntPtr pointer)

  Create a new DisposableIntPtr.
- DisposableIntPtr (IntPtr pointer, long bytesAllocated)

  Create a new DisposableIntPtr, adding memory pressure to the GC to account for the allocation of unmanaged memory.
- void Dispose ()

7.1.1 Detailed Description

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

Definition at line 421 of file MuPDF.cs.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 DisposableIntPtr() [1/2]

```
MuPDFCore.DisposableIntPtr.DisposableIntPtr (
            IntPtr pointer )
```

Create a new DisposableIntPtr.

Parameters

pointer	The pointer that should be freed upon disposing of this object.

Definition at line 437 of file MuPDF.cs.

7.1.2.2 DisposableIntPtr() [2/2]

```
MuPDFCore.DisposableIntPtr.DisposableIntPtr (
            IntPtr pointer,
            long bytesAllocated )
```

Create a new DisposableIntPtr, adding memory pressure to the GC to account for the allocation of unmanaged memory.

Parameters

pointer	The pointer that should be freed upon disposing of this object.
bytesAllocated	The number of bytes that have been allocated, for adding memory pressure.

Definition at line 447 of file MuPDF.cs.

7.1.3 Member Function Documentation

7.1.3.1 Dispose()

```
void MuPDFCore.DisposableIntPtr.Dispose ( )
```

Definition at line 483 of file MuPDF.cs.

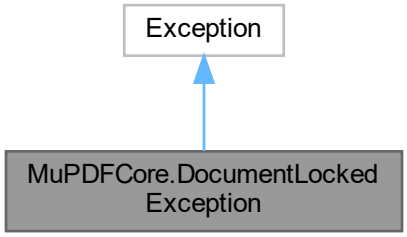
The documentation for this class was generated from the following file:

- MuPDFCore/MuPDF.cs

7.2 MuPDFCore.DocumentLockedException Class Reference

The exception that is thrown when an attempt is made to render an encrypted document without supplying the required password.

Inheritance diagram for MuPDFCore.DocumentLockedException:



7.2.1 Detailed Description

The exception that is thrown when an attempt is made to render an encrypted document without supplying the required password.

Definition at line 509 of file MuPDF.cs.

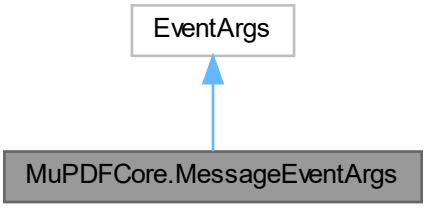
The documentation for this class was generated from the following file:

- MuPDFCore/MuPDF.cs

7.3 MuPDFCore.MessageEventArgs Class Reference

EventArgs for the MuPDF.StandardOutputMessage and MuPDF.StandardErrorMessage events.

Inheritance diagram for MuPDFCore.MessageEventArgs:



Public Member Functions

- MessageEventArgs (string message)

  Create a new MessageEventArgs instance.

Properties

- string Message [get]

  The message that has been logged.

7.3.1 Detailed Description

EventArgs for the MuPDF.StandardOutputMessage and MuPDF.StandardErrorMessage events.

Definition at line 565 of file MuPDF.cs.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 MessageEventArgs()

```
MuPDFCore.MessageEventArgs.MessageEventArgs (
            string message )
```

Create a new MessageEventArgs instance.

Parameters

message	The message that has been logged.

Definition at line 576 of file MuPDF.cs.

7.3.3 Property Documentation

7.3.3.1 Message

```
string MuPDFCore.MessageEventArgs.Message [get]
```

The message that has been logged.

Definition at line 570 of file MuPDF.cs.

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

- MuPDFCore/MuPDF.cs

7.4 MuPDFCore.MuPDF Class Reference

Contains static methods to perform setup operations.

Static Public Member Functions

- static async Task RedirectOutput ()

  Redirects output messages from the native MuPDF library to the StandardOutputMessage and StandardErrorMessage events. Note that this has side-effects.
- static void ResetOutput ()

  Reset the default standard output and error streams for the native MuPDF library.

Events

- static EventHandler< MessageEventArgs > StandardOutputMessage

  This event is invoked when RedirectOutput has been called and the native MuPDF library writes to the standard output stream.
- static EventHandler< MessageEventArgs > StandardErrorMessage

  This event is invoked when RedirectOutput has been called and the native MuPDF library writes to the standard error stream.

7.4.1 Detailed Description

Contains static methods to perform setup operations.

Definition at line 585 of file MuPDF.cs.

7.4.2 Member Function Documentation

7.4.2.1 RedirectOutput()

```
static async Task MuPDFCore.MuPDF.RedirectOutput ( )  [static]
```

Redirects output messages from the native MuPDF library to the StandardOutputMessage and StandardErrorMessage events. Note that this has side-effects.

Returns

A Task that finishes when the output streams have been redirected.

Definition at line 614 of file MuPDF.cs.

7.4.2.2 ResetOutput()

```
static void MuPDFCore.MuPDF.ResetOutput ( )  [static]
```

Reset the default standard output and error streams for the native MuPDF library.

Definition at line 834 of file MuPDF.cs.

7.4.3 Event Documentation

7.4.3.1 StandardErrorMessage

```
EventHandler<MessageEventArgs> MuPDFCore.MuPDF.StandardErrorMessage  [static]
```

This event is invoked when RedirectOutput has been called and the native MuPDF library writes to the standard error stream.

Definition at line 608 of file MuPDF.cs.

7.4.3.2 StandardOutputMessage

```
EventHandler<MessageEventArgs> MuPDFCore.MuPDF.StandardOutputMessage  [static]
```

This event is invoked when RedirectOutput has been called and the native MuPDF library writes to the standard output stream.

Definition at line 603 of file MuPDF.cs.

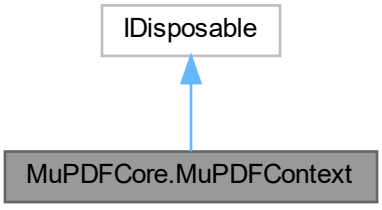
The documentation for this class was generated from the following file:

- MuPDFCore/MuPDF.cs

7.5 MuPDFCore.MuPDFContext Class Reference

A wrapper around a MuPDF context object, which contains the exception stack and the resource cache store.

Inheritance diagram for MuPDFCore.MuPDFContext:



Public Member Functions

- MuPDFContext (uint storeSize=256<< 20)

  Create a new MuPDFContext instance with the specified cache store size.
- void ClearStore ()

  Evict all items from the resource cache store (freeing the memory where they were held).
- void ShrinkStore (double fraction)

  Evict items from the resource cache store (freeing the memory where they were held) until the the size of the store drops to the specified fraction of the current size.
- void Dispose ()

Properties

- long StoreSize `[get]`

  The current size in bytes of the resource cache store. Read-only.
- long StoreMaxSize `[get]`

  The maximum size in bytes of the resource cache store. Read-only.
- int AntiAliasing `[set]`

  Sets the current anti-aliasing level. Changing this value will affect both the TextAntiAliasing and the GraphicsAntiAliasing.
- int TextAntiAliasing `[get, set]`

  Gets or sets the current text anti-aliasing level.
- int GraphicsAntiAliasing `[get, set]`

  Gets or sets the current graphics anti-aliasing level.

7.5.1 Detailed Description

A wrapper around a MuPDF context object, which contains the exception stack and the resource cache store.

Definition at line 25 of file MuPDFContext.cs.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 MuPDFContext()

```
MuPDFCore.MuPDFContext.MuPDFContext (
            uint storeSize = 256 << 20 )
```

Create a new MuPDFContext instance with the specified cache store size.

Parameters

storeSize	The maximum size in bytes of the resource cache store. The default value is 256 MiB.

Definition at line 120 of file MuPDFContext.cs.

7.5.3 Member Function Documentation

7.5.3.1 ClearStore()

```
void MuPDFCore.MuPDFContext.ClearStore ( )
```

Evict all items from the resource cache store (freeing the memory where they were held).

Definition at line 149 of file MuPDFContext.cs.

7.5.3.2 Dispose()

```
void MuPDFCore.MuPDFContext.Dispose ( )
```

Definition at line 194 of file MuPDFContext.cs.

7.5.3.3 ShrinkStore()

```
void MuPDFCore.MuPDFContext.ShrinkStore (
            double fraction )
```

Evict items from the resource cache store (freeing the memory where they were held) until the the size of the store drops to the specified fraction of the current size.

Parameters

fraction	The fraction of the current size that constitutes the target size of the store. If this is <= 0, the cache is cleared. If this is >= 1, nothing happens.

Definition at line 158 of file MuPDFContext.cs.

7.5.4 Property Documentation

7.5.4.1 AntiAliasing

```
int MuPDFCore.MuPDFContext.AntiAliasing [set]
```

Sets the current anti-aliasing level. Changing this value will affect both the TextAntiAliasing and the GraphicsAntiAliasing.

Definition at line 58 of file MuPDFContext.cs.

7.5.4.2 GraphicsAntiAliasing

```
int MuPDFCore.MuPDFContext.GraphicsAntiAliasing  [get], [set]
```

Gets or sets the current graphics anti-aliasing level.

Definition at line 96 of file MuPDFContext.cs.

7.5.4.3 StoreMaxSize

```
long MuPDFCore.MuPDFContext.StoreMaxSize  [get]
```

The maximum size in bytes of the resource cache store. Read-only.

Definition at line 46 of file MuPDFContext.cs.

7.5.4.4 StoreSize

```
long MuPDFCore.MuPDFContext.StoreSize  [get]
```

The current size in bytes of the resource cache store. Read-only.

Definition at line 35 of file MuPDFContext.cs.

7.5.4.5 TextAntiAliasing

```
int MuPDFCore.MuPDFContext.TextAntiAliasing  [get], [set]
```

Gets or sets the current text anti-aliasing level.

Definition at line 74 of file MuPDFContext.cs.

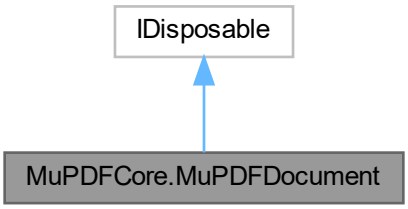
The documentation for this class was generated from the following file:

- MuPDFCore/MuPDFContext.cs

7.6 MuPDFCore.MuPDFDocument Class Reference

A wrapper over a MuPDF document object, which contains possibly multiple pages.

Inheritance diagram for MuPDFCore.MuPDFDocument:



Public Member Functions

- MuPDFDocument (MuPDFContext context, IntPtr dataAddress, long dataLength, InputFileTypes fileType)

  Create a new MuPDFDocument from data bytes accessible through the specified pointer.
- MuPDFDocument (MuPDFContext context, IntPtr dataAddress, long dataLength, InputFileTypes fileType, ref IDisposable dataHolder)

  Create a new MuPDFDocument from data bytes accessible through the specified pointer.
- MuPDFDocument (MuPDFContext context, byte[ ] data, InputFileTypes fileType)

  Create a new MuPDFDocument from an array of bytes.
- MuPDFDocument (MuPDFContext context, ref MemoryStream data, InputFileTypes fileType)

  Create a new MuPDFDocument from a MemoryStream.
- MuPDFDocument (MuPDFContext context, string fileName)

  Create a new MuPDFDocument from a file.
- void ClearCache ()

  Discard all the display lists that have been loaded from the document, possibly freeing some memory in the case of a huge document.
- void Layout (float width, float height, float em)

  Sets the document layout for reflowable document types (e.g., HTML, MOBI). Does not have any effect for documents with a fixed layout (e.g., PDF).
- void LayoutSinglePage (float width, float em)

  Sets the document layout for reflowable document types (e.g., HTML, MOBI), so that the document is rendered to a single page, as tall as necessary. Does not have any effect for documents with a fixed layout (e.g., PDF).
- byte[ ] Render (int pageNumber, Rectangle region, double zoom, PixelFormats pixelFormat, bool include↩ Annotations=true)

  Render (part of) a page to an array of bytes.
- byte[ ] Render (int pageNumber, double zoom, PixelFormats pixelFormat, bool includeAnnotations=true)

  Render a page to an array of bytes.
- void Render (int pageNumber, Rectangle region, double zoom, PixelFormats pixelFormat, IntPtr destination, bool includeAnnotations=true)

  Render (part of) a page to the specified destination.
- void Render (int pageNumber, double zoom, PixelFormats pixelFormat, IntPtr destination, bool include↩ Annotations=true)

Render a page to the specified destination.

- Span< byte > Render (int pageNumber, Rectangle region, double zoom, PixelFormats pixelFormat, out IDisposable disposable, bool includeAnnotations=true)

  Render (part of) a page to a Span<byte>.
- Span< byte > Render (int pageNumber, double zoom, PixelFormats pixelFormat, out IDisposable disposable, bool includeAnnotations=true)

  Render a page to a Span<byte>.
- MuPDFMultiThreadedPageRenderer GetMultiThreadedRenderer (int pageNumber, int threadCount, bool includeAnnotations=true)

  Create a new MuPDFMultiThreadedPageRenderer that renders the specified page with the specified number of threads.
- int GetRenderedSize (int pageNumber, double zoom, PixelFormats pixelFormat)

  Determine how many bytes will be necessary to render the specified page at the specified zoom level, using the the specified pixel format.
- void SaveImage (int pageNumber, Rectangle region, double zoom, PixelFormats pixelFormat, string fileName, RasterOutputFileTypes fileType, bool includeAnnotations=true)

  Save (part of) a page to an image file in the specified format.
- void SaveImageAsJPEG (int pageNumber, Rectangle region, double zoom, string fileName, int quality, bool includeAnnotations=true)

  Save (part of) a page to an image file in JPEG format, with the specified quality.
- void SaveImage (int pageNumber, double zoom, PixelFormats pixelFormat, string fileName, RasterOutputFileTypes fileType, bool includeAnnotations=true)

  Save a page to an image file in the specified format.
- void SaveImageAsJPEG (int pageNumber, double zoom, string fileName, int quality, bool includeAnnotations=true)

  Save a page to an image file in JPEG format, with the specified quality.
- void WriteImage (int pageNumber, Rectangle region, double zoom, PixelFormats pixelFormat, Stream outputStream, RasterOutputFileTypes fileType, bool includeAnnotations=true)

  Write (part of) a page to an image stream in the specified format.
- void WriteImageAsJPEG (int pageNumber, Rectangle region, double zoom, Stream outputStream, int quality, bool includeAnnotations=true)

  Write (part of) a page to an image stream in JPEG format, with the specified quality.
- void WriteImage (int pageNumber, double zoom, PixelFormats pixelFormat, Stream outputStream, RasterOutputFileTypes fileType, bool includeAnnotations=true)

  Write a page to an image stream in the specified format.
- void WriteImageAsJPEG (int pageNumber, double zoom, Stream outputStream, int quality, bool includeAnnotations=true)

  Write a page to an image stream in JPEG format, with the specified quality.
- MuPDFStructuredTextPage GetStructuredTextPage (int pageNumber, bool includeAnnotations=true)

  Creates a new MuPDFStructuredTextPage from the specified page. This contains information about the text layout that can be used for highlighting and searching. The reading order is taken from the order the text is drawn in the source file, so may not be accurate.
- MuPDFStructuredTextPage GetStructuredTextPage (int pageNumber, TesseractLanguage ocrLanguage, bool includeAnnotations=true, CancellationToken cancellationToken=default, IProgress< OCRProgressInfo > progress=null)

  Creates a new MuPDFStructuredTextPage from the specified page, using optical character recognition (OCR) to determine what text is written on the image. This contains information about the text layout that can be used for highlighting and searching.
- async Task< MuPDFStructuredTextPage > GetStructuredTextPageAsync (int pageNumber, TesseractLanguage ocrLanguage, bool includeAnnotations=true, CancellationToken cancellationToken=default, IProgress< OCRProgressInfo > progress=null)

  Creates a new MuPDFStructuredTextPage from the specified page, using optical character recognition (OCR) to determine what text is written on the image. This contains information about the text layout that can be used for highlighting and searching. The OCR step is run asynchronously, e.g. to avoid blocking the UI thread.

- string ExtractText (string separator=null, bool includeAnnotations=true)

  Extracts all the text from the document and returns it as a string. The reading order is taken from the order the text is drawn in the source file, so may not be accurate.
- string ExtractText (TesseractLanguage ocrLanguage, string separator=null, bool includeAnnotations=true)

  Extracts all the text from the document and returns it as a string, using optical character recognition (OCR) to determine what text is written on the image.
- async Task< string > ExtractTextAsync (TesseractLanguage ocrLanguage, string separator=null, bool includeAnnotations=true, CancellationToken cancellationToken=default, IProgress< OCRProgressInfo > progress=null)

  Extracts all the text from the document and returns it as a string, using optical character recognition (OCR) to determine what text is written on the image. The OCR step is run asynchronously, e.g. to avoid blocking the UI thread.
- bool TryUnlock (string password)

  Attempts to unlock the document with the supplied password.
- bool TryUnlock (string password, out PasswordTypes passwordType)

  Attempts to unlock the document with the supplied password.
- void Dispose ()

Static Public Member Functions

- static int GetRenderedSize (Rectangle region, double zoom, PixelFormats pixelFormat)

  Determine how many bytes will be necessary to render the specified region in page units at the specified zoom level, using the the specified pixel format.
- static void CreateDocument (MuPDFContext context, string fileName, DocumentOutputFileTypes fileType, bool includeAnnotations=true, params(MuPDFPage page, Rectangle region, float zoom)[ ] pages)

  Create a new document containing the specified (parts of) pages from other documents.
- static void CreateDocument (MuPDFContext context, string fileName, DocumentOutputFileTypes fileType, bool includeAnnotations=true, params MuPDFPage[ ] pages)

  Create a new document containing the specified pages from other documents.

Properties

- MuPDFPageCollection Pages `[get]`

  The pages contained in the document.
- bool ClipToPageBounds = true `[get, set]`

  Defines whether the images resulting from rendering operations should be clipped to the page boundaries.
- EncryptionState EncryptionState `[get]`

  Describes the encryption state of the document.
- RestrictionState RestrictionState `[get]`

  Describes the restriction state of the document.
- DocumentRestrictions Restrictions `[get]`

  Describes the operations that are restricted on the document. This is not actually enforced by the library, but library users should only allow these operations if the document has been unlocked with the owner password (i.e. if RestrictionState is RestrictionState.Unlocked).

7.6.1 Detailed Description

A wrapper over a MuPDF document object, which contains possibly multiple pages.

Definition at line 30 of file MuPDFDocument.cs.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 MuPDFDocument() [1/5]

```
MuPDFCore.MuPDFDocument.MuPDFDocument (
            MuPDFContext context,
            IntPtr dataAddress,
            long dataLength,
            InputFileTypes fileType )
```

Create a new MuPDFDocument from data bytes accessible through the specified pointer.

Parameters

context	The context that will own this document.
dataAddress	A pointer to the data bytes that make up the document.
dataLength	The number of bytes to read from the specified address.
fileType	The type of the document to read.

Definition at line 137 of file MuPDFDocument.cs.

7.6.2.2 MuPDFDocument() [2/5]

```
MuPDFCore.MuPDFDocument.MuPDFDocument (
            MuPDFContext context,
            IntPtr dataAddress,
            long dataLength,
            InputFileTypes fileType,
            ref IDisposable dataHolder )
```

Create a new MuPDFDocument from data bytes accessible through the specified pointer.

Parameters

context	The context that will own this document.
dataAddress	A pointer to the data bytes that make up the document.
dataLength	The number of bytes to read from the specified address.
fileType	The type of the document to read.
dataHolder	An IDisposable that will be disposed when the MuPDFDocument is disposed.

Definition at line 147 of file MuPDFDocument.cs.

7.6.2.3 MuPDFDocument() [3/5]

```
MuPDFCore.MuPDFDocument.MuPDFDocument (
            MuPDFContext context,
            byte[] data,
            InputFileTypes fileType )
```

Create a new MuPDFDocument from an array of bytes.

Parameters

context	The context that will own this document.
data	An array containing the data bytes that make up the document. This must not be altered until after the MuPDFDocument has been disposed! The address of the array will be pinned, which may cause degradation in the Garbage Collector's performance, and is thus only advised for short-lived documents. To avoid this issue, marshal the bytes to unmanaged memory and use one of the IntPtr constructors.
fileType	The type of the document to read.

Definition at line 247 of file MuPDFDocument.cs.

7.6.2.4 MuPDFDocument() [4/5]

```
MuPDFCore.MuPDFDocument.MuPDFDocument (
            MuPDFContext context,
            ref MemoryStream data,
            InputFileTypes fileType )
```

Create a new MuPDFDocument from a MemoryStream.

Parameters

context	The context that will own this document.
data	The MemoryStream containing the data that makes up the document. This will be disposed when the MuPDFDocument has been disposed and must not be disposed externally! The address of the MemoryStream's buffer will be pinned, which may cause degradation in the Garbage Collector's performance, and is thus only advised for short-lived documents. To avoid this issue, marshal the bytes to unmanaged memory and use one of the IntPtr constructors.
fileType	The type of the document to read.

Definition at line 349 of file MuPDFDocument.cs.

7.6.2.5 MuPDFDocument() [5/5]

```
MuPDFCore.MuPDFDocument.MuPDFDocument (
            MuPDFContext context,
            string fileName )
```

Create a new MuPDFDocument from a file.

Parameters

context	The context that will own this document.
fileName	The path to the file to open.

Definition at line 455 of file MuPDFDocument.cs.

7.6.3 Member Function Documentation

7.6.3.1 ClearCache()

```
void MuPDFCore.MuPDFDocument.ClearCache ( )
```

Discard all the display lists that have been loaded from the document, possibly freeing some memory in the case of a huge document.

Definition at line 586 of file MuPDFDocument.cs.

7.6.3.2 CreateDocument() [1/2]

```
static void MuPDFCore.MuPDFDocument.CreateDocument (
            MuPDFContext context,
            string fileName,
            DocumentOutputFileTypes fileType,
            bool includeAnnotations = true,
            params MuPDFPage[] pages )  [static]
```

Create a new document containing the specified pages from other documents.

Parameters

context	The context that was used to open the documents.
fileName	The output file name.
fileType	The output file format.
pages	The pages to include in the document.
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.

Definition at line 1377 of file MuPDFDocument.cs.

7.6.3.3 CreateDocument() [2/2]

```
static void MuPDFCore.MuPDFDocument.CreateDocument (
```

```
            MuPDFContext context,
            string fileName,
            DocumentOutputFileTypes fileType,
            bool includeAnnotations = true,
            params(MuPDFPage page, Rectangle region, float zoom)[] pages )  [static]
```

Create a new document containing the specified (parts of) pages from other documents.

Parameters

context	The context that was used to open the documents.
fileName	The output file name.
fileType	The output file format.
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.
pages	The pages to include in the document. The "page" element specifies the page, the "region" element the area of the page that should be included in the document, and the "zoom" element how much the region should be scaled.

Definition at line 1269 of file MuPDFDocument.cs.

7.6.3.4 Dispose()

```
void MuPDFCore.MuPDFDocument.Dispose ( )
```

Definition at line 1736 of file MuPDFDocument.cs.

7.6.3.5 ExtractText() [1/2]

```
string MuPDFCore.MuPDFDocument.ExtractText (
            string separator = null,
            bool includeAnnotations = true )
```

Extracts all the text from the document and returns it as a string. The reading order is taken from the order the text is drawn in the source file, so may not be accurate.

Parameters

separator	The character(s) used to separate the text lines obtained from the document. If this is `null`, Environment.NewLine is used as a default separator.
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included. Otherwise, only the page contents are included.

Returns

A string containing all the text in the document. Characters are converted from the UTF-8 representation used in the document to equivalent UTF-16 strings.

Definition at line 1483 of file MuPDFDocument.cs.

7.6.3.6 ExtractText() [2/2]

```
string MuPDFCore.MuPDFDocument.ExtractText (
            TesseractLanguage ocrLanguage,
            string separator = null,
            bool includeAnnotations = true )
```

Extracts all the text from the document and returns it as a string, using optical character recognition (OCR) to determine what text is written on the image.

Parameters

separator	The character(s) used to separate the text lines obtained from the document. If this is `null`, Environment.NewLine is used as a default separator.
ocrLanguage	The language to use for optical character recognition (OCR). If this is null, no OCR is performed.
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included. Otherwise, only the page contents are included.

Returns

A string containing all the text in the document. Characters are converted from the UTF-8 representation used in the document to equivalent UTF-16 strings.

Definition at line 1530 of file MuPDFDocument.cs.

7.6.3.7 ExtractTextAsync()

```
async Task< string > MuPDFCore.MuPDFDocument.ExtractTextAsync (
            TesseractLanguage ocrLanguage,
            string separator = null,
            bool includeAnnotations = true,
            CancellationToken cancellationToken = default,
            IProgress< OCRProgressInfo > progress = null )
```

Extracts all the text from the document and returns it as a string, using optical character recognition (OCR) to determine what text is written on the image. The OCR step is run asynchronously, e.g. to avoid blocking the UI thread.

Parameters

separator	The character(s) used to separate the text lines obtained from the document. If this is `null`, Environment.NewLine is used as a default separator.
ocrLanguage	The language to use for optical character recognition (OCR). If this is null, no OCR is performed.
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included. Otherwise, only the page contents are included.
cancellationToken	A CancellationToken used to cancel the operation.
progress	An IProgress<OCRProgressInfo> used to report progress.

Returns

A string containing all the text in the document. Characters are converted from the UTF-8 representation used in the document to equivalent UTF-16 strings.

Definition at line 1579 of file MuPDFDocument.cs.

7.6.3.8 GetMultiThreadedRenderer()

```
MuPDFMultiThreadedPageRenderer MuPDFCore.MuPDFDocument.GetMultiThreadedRenderer (
            int pageNumber,
            int threadCount,
            bool includeAnnotations = true )
```

Create a new MuPDFMultiThreadedPageRenderer that renders the specified page with the specified number of threads.

Parameters

pageNumber	The number of the page to render (starting at 0).
threadCount	The number of threads to use. This must be factorisable using only powers of 2, 3, 5 or 7. Otherwise, the biggest number smaller than *threadCount* that satisfies this condition is used.

Returns

A MuPDFMultiThreadedPageRenderer that can be used to render the specified page with the specified number of threads.

Parameters

includeAnnotations	If this is `true`, annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.

Definition at line 835 of file MuPDFDocument.cs.

7.6.3.9 GetRenderedSize() [1/2]

```
int MuPDFCore.MuPDFDocument.GetRenderedSize (
            int pageNumber,
            double zoom,
            PixelFormats pixelFormat )
```

Determine how many bytes will be necessary to render the specified page at the specified zoom level, using the the specified pixel format.

Parameters

pageNumber	The number of the page to render (starting at 0).
zoom	The scale at which the page will be rendered. This will determine the size in pixel of the image.
pixelFormat	The format of the pixels data.

Returns

An integer representing the number of bytes that will be necessary to store the pixel data of the rendered image.

Definition at line 857 of file MuPDFDocument.cs.

7.6.3.10 GetRenderedSize() [2/2]

```
static int MuPDFCore.MuPDFDocument.GetRenderedSize (
            Rectangle region,
            double zoom,
            PixelFormats pixelFormat )  [static]
```

Determine how many bytes will be necessary to render the specified region in page units at the specified zoom level, using the the specified pixel format.

Parameters

region	The region that will be rendered.
zoom	The scale at which the region will be rendered. This will determine the size in pixel of the image.
pixelFormat	The format of the pixels data.

Returns

An integer representing the number of bytes that will be necessary to store the pixel data of the rendered image.

Definition at line 874 of file MuPDFDocument.cs.

7.6.3.11 GetStructuredTextPage() [1/2]

```
MuPDFStructuredTextPage MuPDFCore.MuPDFDocument.GetStructuredTextPage (
            int pageNumber,
            bool includeAnnotations = true )
```

Creates a new MuPDFStructuredTextPage from the specified page. This contains information about the text layout that can be used for highlighting and searching. The reading order is taken from the order the text is drawn in the source file, so may not be accurate.

Parameters

pageNumber	The number of the page (starting at 0)
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included. Otherwise, only the page contents are included.

Returns

A MuPDFStructuredTextPage containing a structured text representation of the page.

Definition at line 1396 of file MuPDFDocument.cs.

7.6.3.12 GetStructuredTextPage() [2/2]

```
MuPDFStructuredTextPage MuPDFCore.MuPDFDocument.GetStructuredTextPage (
            int pageNumber,
            TesseractLanguage ocrLanguage,
            bool includeAnnotations = true,
            CancellationToken cancellationToken = default,
            IProgress< OCRProgressInfo > progress = null )
```

Creates a new MuPDFStructuredTextPage from the specified page, using optical character recognition (OCR) to determine what text is written on the image. This contains information about the text layout that can be used for highlighting and searching.

Parameters

pageNumber	The number of the page (starting at 0)
ocrLanguage	The language to use for optical character recognition (OCR). If this is null, no OCR is performed.
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included. Otherwise, only the page contents are included.
cancellationToken	A CancellationToken used to cancel the operation. Providing a value other than the default is not supported on Windows x86 and will throw a runtime exception.
progress	An IProgress<OCRProgressInfo> used to report progress. Providing a value other than null is not supported on Windows x86 and will throw a runtime exception.

Returns

A MuPDFStructuredTextPage containing a structured text representation of the page.

Definition at line 1420 of file MuPDFDocument.cs.

7.6.3.13 GetStructuredTextPageAsync()

```
async Task< MuPDFStructuredTextPage > MuPDFCore.MuPDFDocument.GetStructuredTextPageAsync (
            int pageNumber,
```

```
            TesseractLanguage ocrLanguage,
            bool includeAnnotations = true,
            CancellationToken cancellationToken = default,
            IProgress< OCRProgressInfo > progress = null )
```

Creates a new MuPDFStructuredTextPage from the specified page, using optical character recognition (OCR) to determine what text is written on the image. This contains information about the text layout that can be used for highlighting and searching. The OCR step is run asynchronously, e.g. to avoid blocking the UI thread.

Parameters

pageNumber	The number of the page (starting at 0)
ocrLanguage	The language to use for optical character recognition (OCR). If this is null, no OCR is performed.
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included. Otherwise, only the page contents are included.
cancellationToken	A CancellationToken used to cancel the operation. Providing a value other than the default is not supported on Windows x86 and will throw a runtime exception.
progress	An IProgress<OCRProgressInfo> used to report progress. Providing a value other than null is not supported on Windows x86 and will throw a runtime exception.

Returns

A MuPDFStructuredTextPage containing a structured text representation of the page.

Definition at line 1453 of file MuPDFDocument.cs.

7.6.3.14 Layout()

```
void MuPDFCore.MuPDFDocument.Layout (
            float width,
            float height,
            float em )
```

Sets the document layout for reflowable document types (e.g., HTML, MOBI). Does not have any effect for documents with a fixed layout (e.g., PDF).

Parameters

width	The width of each page, in points. Must be > 0.
height	The height of each page, in points. Must be > 0.
em	The default font size, in points.

Definition at line 601 of file MuPDFDocument.cs.

7.6.3.15 LayoutSinglePage()

```
void MuPDFCore.MuPDFDocument.LayoutSinglePage (
```

```
            float width,
            float em )
```

Sets the document layout for reflowable document types (e.g., HTML, MOBI), so that the document is rendered to a single page, as tall as necessary. Does not have any effect for documents with a fixed layout (e.g., PDF).

Parameters

width	The width of each page, in points. Must be > 0.
em	The default font size, in points.

Definition at line 629 of file MuPDFDocument.cs.

7.6.3.16 Render() [1/6]

```
byte[] MuPDFCore.MuPDFDocument.Render (
            int pageNumber,
            double zoom,
            PixelFormats pixelFormat,
            bool includeAnnotations = true )
```

Render a page to an array of bytes.

Parameters

pageNumber	The number of the page to render (starting at 0).
zoom	The scale at which the page will be rendered. This will determine the size in pixel of the image.
pixelFormat	The format of the pixel data.
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.

Returns

A byte array containing the raw values for the pixels of the rendered image.

Definition at line 689 of file MuPDFDocument.cs.

7.6.3.17 Render() [2/6]

```
void MuPDFCore.MuPDFDocument.Render (
            int pageNumber,
            double zoom,
            PixelFormats pixelFormat,
            IntPtr destination,
            bool includeAnnotations = true )
```

Render a page to the specified destination.

Parameters

pageNumber	The number of the page to render (starting at 0).
zoom	The scale at which the page will be rendered. This will determine the size in pixel of the image.
pixelFormat	The format of the pixel data.
destination	The address of the buffer where the pixel data will be written. There must be enough space available to write the values for all the pixels, otherwise this will fail catastrophically!
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.

Definition at line 768 of file MuPDFDocument.cs.

7.6.3.18 Render() [3/6]

```
Span< byte > MuPDFCore.MuPDFDocument.Render (
            int pageNumber,
            double zoom,
            PixelFormats pixelFormat,
            out IDisposable disposable,
            bool includeAnnotations = true )
```

Render a page to a Span<byte>.

Parameters

pageNumber	The number of the page to render (starting at 0).
zoom	The scale at which the page will be rendered. This will determine the size in pixel of the image.
pixelFormat	The format of the pixel data.
disposable	An IDisposable that can be used to free the memory where the image is stored. You should keep track of this and dispose it when you have finished working with the image.
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.

Definition at line 816 of file MuPDFDocument.cs.

7.6.3.19 Render() [4/6]

```
byte[] MuPDFCore.MuPDFDocument.Render (
            int pageNumber,
            Rectangle region,
            double zoom,
            PixelFormats pixelFormat,
            bool includeAnnotations = true )
```

Render (part of) a page to an array of bytes.

Parameters

pageNumber	The number of the page to render (starting at 0).
region	The region of the page to render in page units.
zoom	The scale at which the page will be rendered. This will determine the size in pixel of the image.
pixelFormat	The format of the pixel data.
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.

Returns

A byte array containing the raw values for the pixels of the rendered image.

Definition at line 655 of file MuPDFDocument.cs.

7.6.3.20 Render() [5/6]

```
void MuPDFCore.MuPDFDocument.Render (
            int pageNumber,
            Rectangle region,
            double zoom,
            PixelFormats pixelFormat,
            IntPtr destination,
            bool includeAnnotations = true )
```

Render (part of) a page to the specified destination.

Parameters

pageNumber	The number of the page to render (starting at 0).
region	The region of the page to render in page units.
zoom	The scale at which the page will be rendered. This will determine the size in pixel of the image.
pixelFormat	The format of the pixel data.
destination	The address of the buffer where the pixel data will be written. There must be enough space available to write the values for all the pixels, otherwise this will fail catastrophically!
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.

Definition at line 709 of file MuPDFDocument.cs.

7.6.3.21 Render() [6/6]

```
Span< byte > MuPDFCore.MuPDFDocument.Render (
            int pageNumber,
```

```
            Rectangle region,
            double zoom,
            PixelFormats pixelFormat,
            out IDisposable disposable,
            bool includeAnnotations = true )
```

Render (part of) a page to a Span<byte>.

Parameters

pageNumber	The number of the page to render (starting at 0).
region	The region of the page to render in page units.
zoom	The scale at which the page will be rendered. This will determine the size in pixel of the image.
pixelFormat	The format of the pixel data.
disposable	An IDisposable that can be used to free the memory where the image is stored. You should keep track of this and dispose it when you have finished working with the image.
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.

Definition at line 788 of file MuPDFDocument.cs.

7.6.3.22 SaveImage() [1/2]

```
void MuPDFCore.MuPDFDocument.SaveImage (
            int pageNumber,
            double zoom,
            PixelFormats pixelFormat,
            string fileName,
            RasterOutputFileTypes fileType,
            bool includeAnnotations = true )
```

Save a page to an image file in the specified format.

Parameters

pageNumber	The number of the page to render (starting at 0).
zoom	The scale at which the page will be rendered. This will determine the size in pixel of the image.
pixelFormat	The format of the pixel data.
fileName	The path to the output file.
fileType	The output format of the file.
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.

Definition at line 1038 of file MuPDFDocument.cs.

7.6.3.23 SaveImage() [2/2]

```
void MuPDFCore.MuPDFDocument.SaveImage (
            int pageNumber,
            Rectangle region,
            double zoom,
            PixelFormats pixelFormat,
            string fileName,
            RasterOutputFileTypes fileType,
            bool includeAnnotations = true )
```

Save (part of) a page to an image file in the specified format.

Parameters

pageNumber	The number of the page to render (starting at 0).
region	The region of the page to render in page units.
zoom	The scale at which the page will be rendered. This will determine the size in pixel of the image.
pixelFormat	The format of the pixel data.
fileName	The path to the output file.
fileType	The output format of the file.
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.

Definition at line 910 of file MuPDFDocument.cs.

7.6.3.24 SaveImageAsJPEG() [1/2]

```
void MuPDFCore.MuPDFDocument.SaveImageAsJPEG (
            int pageNumber,
            double zoom,
            string fileName,
            int quality,
            bool includeAnnotations = true )
```

Save a page to an image file in JPEG format, with the specified quality.

Parameters

pageNumber	The number of the page to render (starting at 0).
zoom	The scale at which the page will be rendered. This will determine the size in pixel of the image.
fileName	The path to the output file.
quality	The quality of the JPEG output file (ranging from 0 to 100).
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.

Definition at line 1057 of file MuPDFDocument.cs.

7.6.3.25 SaveImageAsJPEG() [2/2]

```
void MuPDFCore.MuPDFDocument.SaveImageAsJPEG (
            int pageNumber,
            Rectangle region,
            double zoom,
            string fileName,
            int quality,
            bool includeAnnotations = true )
```

Save (part of) a page to an image file in JPEG format, with the specified quality.

Parameters

pageNumber	The number of the page to render (starting at 0).
region	The region of the page to render in page units.
zoom	The scale at which the page will be rendered. This will determine the size in pixel of the image.
fileName	The path to the output file.
quality	The quality of the JPEG output file (ranging from 0 to 100).
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.

Definition at line 979 of file MuPDFDocument.cs.

7.6.3.26 TryUnlock() [1/2]

```
bool MuPDFCore.MuPDFDocument.TryUnlock (
            string password )
```

Attempts to unlock the document with the supplied password.

Parameters

password	The user or owner password to use to unlock the document.

Returns

`true`If the document was successfully unlocked (or if it was never locked to begin with), `false` if the password was incorrect and the document is still locked.

This method can be used both to unlock an encrypted document and to check whether the supplied owner password is correct.

Definition at line 1625 of file MuPDFDocument.cs.

7.6.3.27 TryUnlock() [2/2]

```
bool MuPDFCore.MuPDFDocument.TryUnlock (
            string password,
            out PasswordTypes passwordType )
```

Attempts to unlock the document with the supplied password.

Parameters

password	The user or owner password to use to unlock the document.
passwordType	If the method returns true, this can be used to determine whether the supplied password was the user password or the owner password. If the method returns `false`, this can be used to determine whether a user password and/or an owner password are required.

Returns

`true`If the document was successfully unlocked (or if it was never locked to begin with), `false` if the password was incorrect and the document is still locked.

This method can be used both to unlock an encrypted document and to check whether the supplied owner password is correct.

Definition at line 1638 of file MuPDFDocument.cs.

7.6.3.28 WriteImage() [1/2]

```
void MuPDFCore.MuPDFDocument.WriteImage (
            int pageNumber,
            double zoom,
            PixelFormats pixelFormat,
            Stream outputStream,
            RasterOutputFileTypes fileType,
            bool includeAnnotations = true )
```

Write a page to an image stream in the specified format.

Parameters

pageNumber	The number of the page to render (starting at 0).
zoom	The scale at which the page will be rendered. This will determine the size in pixel of the image.
pixelFormat	The format of the pixel data.
outputStream	The stream to which the image data will be written.
fileType	The output format of the image.
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.

Definition at line 1231 of file MuPDFDocument.cs.

7.6.3.29 WriteImage() [2/2]

```
void MuPDFCore.MuPDFDocument.WriteImage (
            int pageNumber,
            Rectangle region,
            double zoom,
            PixelFormats pixelFormat,
            Stream outputStream,
            RasterOutputFileTypes fileType,
            bool includeAnnotations = true )
```

Write (part of) a page to an image stream in the specified format.

Parameters

pageNumber	The number of the page to render (starting at 0).
region	The region of the page to render in page units.
zoom	The scale at which the page will be rendered. This will determine the size in pixel of the image.
pixelFormat	The format of the pixel data.
outputStream	The stream to which the image data will be written.
fileType	The output format of the image.
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.

Definition at line 1078 of file MuPDFDocument.cs.

7.6.3.30 WriteImageAsJPEG() [1/2]

```
void MuPDFCore.MuPDFDocument.WriteImageAsJPEG (
            int pageNumber,
            double zoom,
            Stream outputStream,
            int quality,
            bool includeAnnotations = true )
```

Write a page to an image stream in JPEG format, with the specified quality.

Parameters

pageNumber	The number of the page to render (starting at 0).
zoom	The scale at which the page will be rendered. This will determine the size in pixel of the image.
outputStream	The stream to which the image data will be written.
quality	The quality of the JPEG output (ranging from 0 to 100).
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.

Definition at line 1250 of file MuPDFDocument.cs.

7.6.3.31 WriteImageAsJPEG() [2/2]

```
void MuPDFCore.MuPDFDocument.WriteImageAsJPEG (
            int pageNumber,
            Rectangle region,
            double zoom,
            Stream outputStream,
            int quality,
            bool includeAnnotations = true )
```

Write (part of) a page to an image stream in JPEG format, with the specified quality.

Parameters

pageNumber	The number of the page to render (starting at 0).
region	The region of the page to render in page units.
zoom	The scale at which the page will be rendered. This will determine the size in pixel of the image.
outputStream	The stream to which the image data will be written.
quality	The quality of the JPEG output (ranging from 0 to 100).
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.

Definition at line 1160 of file MuPDFDocument.cs.

7.6.4 Property Documentation

7.6.4.1 ClipToPageBounds

```
bool MuPDFCore.MuPDFDocument.ClipToPageBounds = true  [get], [set]
```

Defines whether the images resulting from rendering operations should be clipped to the page boundaries.

Definition at line 111 of file MuPDFDocument.cs.

7.6.4.2 EncryptionState

```
EncryptionState MuPDFCore.MuPDFDocument.EncryptionState  [get]
```

Describes the encryption state of the document.

Definition at line 116 of file MuPDFDocument.cs.

7.6.4.3 Pages

```
MuPDFPageCollection MuPDFCore.MuPDFDocument.Pages  [get]
```

The pages contained in the document.

Definition at line 106 of file MuPDFDocument.cs.

7.6.4.4 Restrictions

```
DocumentRestrictions MuPDFCore.MuPDFDocument.Restrictions  [get]
```

Describes the operations that are restricted on the document. This is not actually enforced by the library, but library users should only allow these operations if the document has been unlocked with the owner password (i.e. if RestrictionState is RestrictionState.Unlocked).

Definition at line 128 of file MuPDFDocument.cs.

7.6.4.5 RestrictionState

```
RestrictionState MuPDFCore.MuPDFDocument.RestrictionState  [get]
```

Describes the restriction state of the document.

Definition at line 121 of file MuPDFDocument.cs.

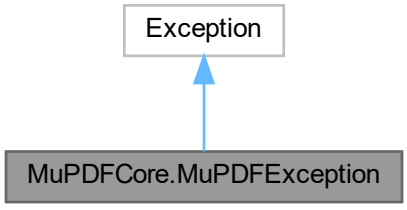
The documentation for this class was generated from the following file:

- MuPDFCore/MuPDFDocument.cs

7.7 MuPDFCore.MuPDFException Class Reference

The exception that is thrown when a MuPDF operation fails.

Inheritance diagram for MuPDFCore.MuPDFException:



Public Attributes

- readonly ExitCodes ErrorCode

  The ExitCodes returned by the native function.

7.7.1 Detailed Description

The exception that is thrown when a MuPDF operation fails.

Definition at line 493 of file MuPDF.cs.

7.7.2 Member Data Documentation

7.7.2.1 ErrorCode

```
readonly ExitCodes MuPDFCore.MuPDFException.ErrorCode
```

The ExitCodes returned by the native function.

Definition at line 498 of file MuPDF.cs.

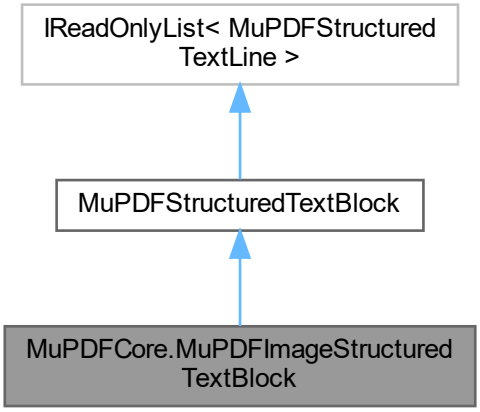
The documentation for this class was generated from the following file:

- MuPDFCore/MuPDF.cs

7.8 MuPDFCore.MuPDFImageStructuredTextBlock Class Reference

Represents a block containing a single image. The block contains a single line with a single character.

Inheritance diagram for MuPDFCore.MuPDFImageStructuredTextBlock:



Public Member Functions

- override IEnumerator< MuPDFStructuredTextLine > GetEnumerator ()

Properties

- override Types Type [get]
- override int Count [get]
- override MuPDFStructuredTextLine this[int index] [get]

Additional Inherited Members

7.8.1 Detailed Description

Represents a block containing a single image. The block contains a single line with a single character.

Definition at line 615 of file MuPDFStructuredTextPage.cs.

7.8.2 Member Function Documentation

7.8.2.1 GetEnumerator()

```
override IEnumerator< MuPDFStructuredTextLine > MuPDFCore.MuPDFImageStructuredTextBlock.Get↩
Enumerator ( ) [virtual]
```

Implements MuPDFCore.MuPDFStructuredTextBlock.

Definition at line 647 of file MuPDFStructuredTextPage.cs.

7.8.3 Property Documentation

7.8.3.1 Count

```
override int MuPDFCore.MuPDFImageStructuredTextBlock.Count [get]
```

Definition at line 621 of file MuPDFStructuredTextPage.cs.

7.8.3.2 this[int index]

```
override MuPDFStructuredTextLine MuPDFCore.MuPDFImageStructuredTextBlock.this[int index]  [get]
```

Definition at line 626 of file MuPDFStructuredTextPage.cs.

7.8.3.3 Type

```
override Types MuPDFCore.MuPDFImageStructuredTextBlock.Type  [get]
```

Definition at line 618 of file MuPDFStructuredTextPage.cs.

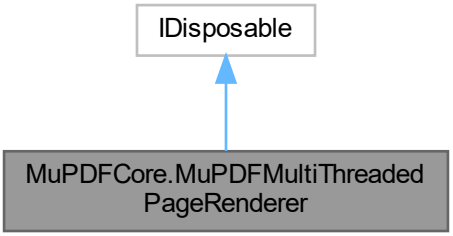
The documentation for this class was generated from the following file:

- MuPDFCore/MuPDFStructuredTextPage.cs

7.9 MuPDFCore.MuPDFMultiThreadedPageRenderer Class Reference

A class that holds the necessary resources to render a page of a MuPDF document using multiple threads.

Inheritance diagram for MuPDFCore.MuPDFMultiThreadedPageRenderer:



Public Member Functions

- void Render (RoundedSize targetSize, Rectangle region, IntPtr[ ] destinations, PixelFormats pixelFormat)

  Render the specified region to an image of the specified size, split in a number of tiles equal to the number of threads used by this MuPDFMultiThreadedPageRenderer, without marshaling. This method will not return until all the rendering threads have finished.
- delegate Span< byte > GetSpanItem (int index)

  Gets an element from a collection of Span<byte>
- GetSpanItem Render (RoundedSize targetSize, Rectangle region, out IDisposable[ ] disposables, PixelFormats pixelFormat)

  Render the specified region to an image of the specified size, split in a number of tiles equal to the number of threads used by this MuPDFMultiThreadedPageRenderer, without marshaling. This method will not return until all the rendering threads have finished. Since creating an array of Span<T> is not allowed, this method returns a delegate that accepts an integer parameter (representing the index of the span in the "array") and returns the Span<T> corresponding to that index.
- void Abort ()

  Signal to the rendering threads that they should abort rendering as soon as possible.
- RenderProgress GetProgress ()

  Get the current rendering progress of all the threads.
- void Dispose ()

Properties

- int ThreadCount [get]

  The number of threads that are used to render the image.

7.9.1 Detailed Description

A class that holds the necessary resources to render a page of a MuPDF document using multiple threads.

Definition at line 276 of file MuPDFMultiThreadedPageRenderer.cs.

7.9.2 Member Function Documentation

7.9.2.1 Abort()

```
void MuPDFCore.MuPDFMultiThreadedPageRenderer.Abort ( )
```

Signal to the rendering threads that they should abort rendering as soon as possible.

Definition at line 543 of file MuPDFMultiThreadedPageRenderer.cs.

7.9.2.2 Dispose()

```
void MuPDFCore.MuPDFMultiThreadedPageRenderer.Dispose ( )
```

Definition at line 592 of file MuPDFMultiThreadedPageRenderer.cs.

7.9.2.3 GetProgress()

```
RenderProgress MuPDFCore.MuPDFMultiThreadedPageRenderer.GetProgress ( )
```

Get the current rendering progress of all the threads.

Returns

A RenderProgress object containing the rendering progress of all the threads.

Definition at line 555 of file MuPDFMultiThreadedPageRenderer.cs.

7.9.2.4 GetSpanItem()

```
delegate Span< byte > MuPDFCore.MuPDFMultiThreadedPageRenderer.GetSpanItem (
            int index )
```

Gets an element from a collection of Span<byte>

Parameters

index	The index of the element to get.

Returns

An element from a collection of Span<byte>

7.9.2.5 Render() [1/2]

```
void MuPDFCore.MuPDFMultiThreadedPageRenderer.Render (
            RoundedSize targetSize,
            Rectangle region,
            IntPtr[] destinations,
            PixelFormats pixelFormat )
```

Render the specified region to an image of the specified size, split in a number of tiles equal to the number of threads used by this MuPDFMultiThreadedPageRenderer, without marshaling. This method will not return until all the rendering threads have finished.

Parameters

targetSize	The total size of the image that should be rendered.
region	The region in page units that should be rendered.
destinations	An array containing the addresses of the buffers where the rendered tiles will be written. There must be enough space available in each buffer to write the values for all the pixels of the tile, otherwise this will fail catastrophically! As long as the *targetSize* is the same, the size in pixel of the tiles is guaranteed to also be the same.
pixelFormat	The format of the pixel data.

Definition at line 383 of file MuPDFMultiThreadedPageRenderer.cs.

7.9.2.6 Render() [2/2]

```
GetSpanItem MuPDFCore.MuPDFMultiThreadedPageRenderer.Render (
            RoundedSize targetSize,
            Rectangle region,
            out IDisposable[] disposables,
            PixelFormats pixelFormat )
```

Render the specified region to an image of the specified size, split in a number of tiles equal to the number of threads used by this MuPDFMultiThreadedPageRenderer, without marshaling. This method will not return until all the rendering threads have finished. Since creating an array of Span<T> is not allowed, this method returns a delegate that accepts an integer parameter (representing the index of the span in the "array") and returns the Span<T> corresponding to that index.

Parameters

targetSize	The total size of the image that should be rendered.
region	The region in page units that should be rendered.
disposables	A collection of IDisposables that can be used to free the memory where the rendered tiles are stored. You should keep track of these and dispose them when you have finished working with the images.
pixelFormat	The format of the pixel data.

Returns

A delegate that accepts an integer parameter (representing the index of the span in the "array") and returns the Span<T> corresponding to that index.

Definition at line 509 of file MuPDFMultiThreadedPageRenderer.cs.

7.9.3 Property Documentation

7.9.3.1 ThreadCount

```
int MuPDFCore.MuPDFMultiThreadedPageRenderer.ThreadCount  [get]
```

The number of threads that are used to render the image.

Definition at line 306 of file MuPDFMultiThreadedPageRenderer.cs.

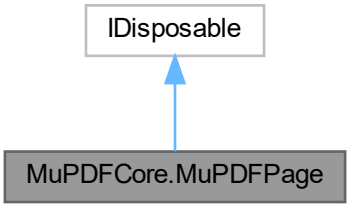
The documentation for this class was generated from the following file:

- MuPDFCore/MuPDFMultiThreadedPageRenderer.cs

7.10 MuPDFCore.MuPDFPage Class Reference

A wrapper over a MuPDF page object, which contains information about the page's boundaries.

Inheritance diagram for MuPDFCore.MuPDFPage:



Public Member Functions

- void Dispose ()

Properties

- Rectangle Bounds [get]

  The page's bounds at 72 DPI. Read-only.
- int PageNumber [get]

  The number of this page in the original document.

7.10.1 Detailed Description

A wrapper over a MuPDF page object, which contains information about the page's boundaries.

Definition at line 27 of file MuPDFPage.cs.

7.10.2 Member Function Documentation

7.10.2.1 Dispose()

```
void MuPDFCore.MuPDFPage.Dispose ( )
```

Definition at line 121 of file MuPDFPage.cs.

7.10.3 Property Documentation

7.10.3.1 Bounds

```
Rectangle MuPDFCore.MuPDFPage.Bounds [get]
```

The page's bounds at 72 DPI. Read-only.

Definition at line 32 of file MuPDFPage.cs.

7.10.3.2 PageNumber

```
int MuPDFCore.MuPDFPage.PageNumber  [get]
```

The number of this page in the original document.

Definition at line 37 of file MuPDFPage.cs.

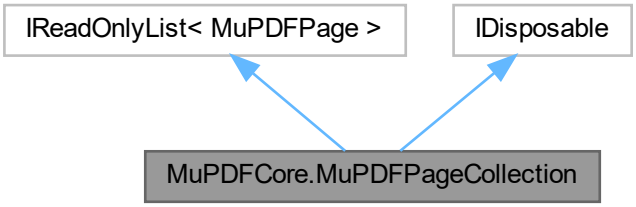
The documentation for this class was generated from the following file:

- MuPDFCore/MuPDFPage.cs

7.11 MuPDFCore.MuPDFPageCollection Class Reference

A lazy collection of MuPDFPages. Each page is loaded from the document as it is requested for the first time.

Inheritance diagram for MuPDFCore.MuPDFPageCollection:



Public Member Functions

- IEnumerator< MuPDFPage > GetEnumerator ()
  inheritdoc/>
- void Dispose ()

Properties

- int Length [get]
  The number of pages in the collection.
- int Count [get]
  The number of pages in the collection.
- MuPDFPage this[int index] [get]
  Get a page from the collection.

7.11.1 Detailed Description

A lazy collection of MuPDFPages. Each page is loaded from the document as it is requested for the first time.

Definition at line 131 of file MuPDFPage.cs.

7.11.2 Member Function Documentation

7.11.2.1 Dispose()

```
void MuPDFCore.MuPDFPageCollection.Dispose ( )
```

Definition at line 234 of file MuPDFPage.cs.

7.11.2.2 GetEnumerator()

```
IEnumerator< MuPDFPage > MuPDFCore.MuPDFPageCollection.GetEnumerator ( )
```

inheritdoc/>

Definition at line 195 of file MuPDFPage.cs.

7.11.3 Property Documentation

7.11.3.1 Count

```
int MuPDFCore.MuPDFPageCollection.Count  [get]
```

The number of pages in the collection.

Definition at line 156 of file MuPDFPage.cs.

7.11.3.2 Length

```
int MuPDFCore.MuPDFPageCollection.Length  [get]
```

The number of pages in the collection.

Definition at line 151 of file MuPDFPage.cs.

7.11.3.3 this[int index]

```
MuPDFPage MuPDFCore.MuPDFPageCollection.this[int index]  [get]
```

Get a page from the collection.

Parameters

index	The number of the page (starting at 0).

Returns

The specified MuPDFPage.

Definition at line 163 of file MuPDFPage.cs.

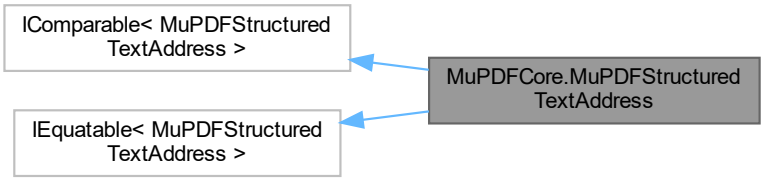
The documentation for this class was generated from the following file:

- MuPDFCore/MuPDFPage.cs

7.12 MuPDFCore.MuPDFStructuredTextAddress Struct Reference

Represents the address of a particular character in a MuPDFStructuredTextPage, in terms of block index, line index and character index.

Inheritance diagram for MuPDFCore.MuPDFStructuredTextAddress:



Public Member Functions

- MuPDFStructuredTextAddress (int blockIndex, int lineIndex, int characterIndex)

  Creates a new MuPDFStructuredTextAddress from the specified indices.
- int CompareTo (MuPDFStructuredTextAddress other)

  Compares this MuPDFStructuredTextAddress with another MuPDFStructuredTextAddress.
- override int GetHashCode ()
- MuPDFStructuredTextAddress? Increment (MuPDFStructuredTextPage page)

  Returns a MuPDFStructuredTextAddress corresponding to the next character in the specified page.
- bool Equals (MuPDFStructuredTextAddress other)

  Compares the current MuPDFStructuredTextAddress with another MuPDFStructuredTextAddress.
- override bool Equals (object other)

Static Public Member Functions

- static bool operator> (MuPDFStructuredTextAddress first, MuPDFStructuredTextAddress second)

  Compares two MuPDFStructuredTextAddress.
- static bool operator>= (MuPDFStructuredTextAddress first, MuPDFStructuredTextAddress second)

  Compares two MuPDFStructuredTextAddress.
- static bool operator< (MuPDFStructuredTextAddress first, MuPDFStructuredTextAddress second)

  Compares two MuPDFStructuredTextAddress.
- static bool operator<= (MuPDFStructuredTextAddress first, MuPDFStructuredTextAddress second)

  Compares two MuPDFStructuredTextAddress.
- static bool operator== (MuPDFStructuredTextAddress first, MuPDFStructuredTextAddress second)

  Compares two MuPDFStructuredTextAddress.
- static bool operator!= (MuPDFStructuredTextAddress first, MuPDFStructuredTextAddress second)

  Compares two MuPDFStructuredTextAddress.

Public Attributes

- readonly int BlockIndex

  The index of the block.
- readonly int LineIndex

  The index of the line within the block.
- readonly int CharacterIndex

  The index of the character within the line.

7.12.1 Detailed Description

Represents the address of a particular character in a MuPDFStructuredTextPage, in terms of block index, line index and character index.

Definition at line 961 of file MuPDFStructuredTextPage.cs.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 MuPDFStructuredTextAddress()

```
MuPDFCore.MuPDFStructuredTextAddress.MuPDFStructuredTextAddress (
            int blockIndex,
            int lineIndex,
            int characterIndex )
```

Creates a new MuPDFStructuredTextAddress from the specified indices.

Parameters

blockIndex	The index of the block.
lineIndex	The index of the line within the block.
characterIndex	The index of the character within the line.

Definition at line 984 of file MuPDFStructuredTextPage.cs.

7.12.3 Member Function Documentation

7.12.3.1 CompareTo()

```
int MuPDFCore.MuPDFStructuredTextAddress.CompareTo (
            MuPDFStructuredTextAddress other )
```

Compares this MuPDFStructuredTextAddress with another MuPDFStructuredTextAddress.

Parameters

other	The MuPDFStructuredTextAddress to compare with the current instance.

Returns

-1 if the *other* MuPDFStructuredTextAddress comes after the current instance, 1 if it comes before, or 0 if they represent the same address.

Definition at line 996 of file MuPDFStructuredTextPage.cs.

7.12.3.2 Equals() [1/2]

```
bool MuPDFCore.MuPDFStructuredTextAddress.Equals (
            MuPDFStructuredTextAddress other )
```

Compares the current MuPDFStructuredTextAddress with another MuPDFStructuredTextAddress.

Parameters

other	The other MuPDFStructuredTextAddress to compare with the current instance.

Returns

`true` if the two MuPDFStructuredTextAddresses represent the same address; otherwise, `false`.

Definition at line 1239 of file MuPDFStructuredTextPage.cs.

7.12.3.3 Equals() [2/2]

```
override bool MuPDFCore.MuPDFStructuredTextAddress.Equals (
            object other )
```

Definition at line 1245 of file MuPDFStructuredTextPage.cs.

7.12.3.4 GetHashCode()

```
override int MuPDFCore.MuPDFStructuredTextAddress.GetHashCode ( )
```

Definition at line 1195 of file MuPDFStructuredTextPage.cs.

7.12.3.5 Increment()

```
MuPDFStructuredTextAddress?  MuPDFCore.MuPDFStructuredTextAddress.Increment (
            MuPDFStructuredTextPage page )
```

Returns a MuPDFStructuredTextAddress corresponding to the next character in the specified page.

Parameters

page	The page the address refers to.

Returns

A MuPDFStructuredTextAddress corresponding to the next character in the specified page.

Definition at line 1208 of file MuPDFStructuredTextPage.cs.

7.12.3.6 operator"!=()

```
static bool MuPDFCore.MuPDFStructuredTextAddress.operator!= (
            MuPDFStructuredTextAddress first,
            MuPDFStructuredTextAddress second )  [static]
```

Compares two MuPDFStructuredTextAddress.

Parameters

first	The first MuPDFStructuredTextAddress to compare.
second	The second MuPDFStructuredTextAddress to compare.

Returns

`true` if the two MuPDFStructuredTextAddresses represent different addresses; otherwise, `false`.

Definition at line 1189 of file MuPDFStructuredTextPage.cs.

7.12.3.7 operator<()

```
static bool MuPDFCore.MuPDFStructuredTextAddress.operator< (
            MuPDFStructuredTextAddress first,
            MuPDFStructuredTextAddress second )  [static]
```

Compares two MuPDFStructuredTextAddress.

Parameters

first	The first MuPDFStructuredTextAddress to compare.
second	The second MuPDFStructuredTextAddress to compare.

Returns

`true` if the *first* MuPDFStructuredTextAddress comes before the *second* one; otherwise, `false`.

Definition at line 1098 of file MuPDFStructuredTextPage.cs.

7.12.3.8 operator<=()

```
static bool MuPDFCore.MuPDFStructuredTextAddress.operator<= (
            MuPDFStructuredTextAddress first,
            MuPDFStructuredTextAddress second )  [static]
```

Compares two MuPDFStructuredTextAddress.

Parameters

first	The first MuPDFStructuredTextAddress to compare.
second	The second MuPDFStructuredTextAddress to compare.

Returns

`true` if the *first* MuPDFStructuredTextAddress comes before the *second* one or if they represent the same address; otherwise, `false`.

Definition at line 1138 of file MuPDFStructuredTextPage.cs.

7.12.3.9 operator==()

```
static bool MuPDFCore.MuPDFStructuredTextAddress.operator== (
            MuPDFStructuredTextAddress first,
            MuPDFStructuredTextAddress second )  [static]
```

Compares two MuPDFStructuredTextAddress.

Parameters

first	The first MuPDFStructuredTextAddress to compare.
second	The second MuPDFStructuredTextAddress to compare.

Returns

`true` if the two MuPDFStructuredTextAddresses represent the same address; otherwise, `false`.

Definition at line 1178 of file MuPDFStructuredTextPage.cs.

7.12.3.10 operator>()

```
static bool MuPDFCore.MuPDFStructuredTextAddress.operator> (
            MuPDFStructuredTextAddress first,
            MuPDFStructuredTextAddress second )  [static]
```

Compares two MuPDFStructuredTextAddress.

Parameters

first	The first MuPDFStructuredTextAddress to compare.
second	The second MuPDFStructuredTextAddress to compare.

Returns

`true` if the *first* MuPDFStructuredTextAddress comes after the *second* one; otherwise, `false`.

Definition at line 1018 of file MuPDFStructuredTextPage.cs.

7.12.3.11 operator>=()

```
static bool MuPDFCore.MuPDFStructuredTextAddress.operator>= (
            MuPDFStructuredTextAddress first,
            MuPDFStructuredTextAddress second )  [static]
```

Compares two MuPDFStructuredTextAddress.

Parameters

first	The first MuPDFStructuredTextAddress to compare.
second	The second MuPDFStructuredTextAddress to compare.

Returns

`true` if the *first* MuPDFStructuredTextAddress comes after the *second* one or if they represent the same address; otherwise, `false`.

Definition at line 1058 of file MuPDFStructuredTextPage.cs.

7.12.4 Member Data Documentation

7.12.4.1 BlockIndex

```
readonly int MuPDFCore.MuPDFStructuredTextAddress.BlockIndex
```

The index of the block.

Definition at line 966 of file MuPDFStructuredTextPage.cs.

7.12.4.2 CharacterIndex

```
readonly int MuPDFCore.MuPDFStructuredTextAddress.CharacterIndex
```

The index of the character within the line.

Definition at line 976 of file MuPDFStructuredTextPage.cs.

7.12.4.3 LineIndex

```
readonly int MuPDFCore.MuPDFStructuredTextAddress.LineIndex
```

The index of the line within the block.

Definition at line 971 of file MuPDFStructuredTextPage.cs.

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

- MuPDFCore/MuPDFStructuredTextPage.cs

7.13 MuPDFCore.MuPDFStructuredTextAddressSpan Class Reference

Represents a range of characters in a MuPDFStructuredTextPage.

Public Member Functions

- MuPDFStructuredTextAddressSpan (MuPDFStructuredTextAddress start, MuPDFStructuredTextAddress? end)

  Creates a new MuPDFStructuredTextAddressSpan corresponding to the specified character range.

Public Attributes

- readonly MuPDFStructuredTextAddress Start

  The addres of the start of the range.
- readonly? MuPDFStructuredTextAddress End

  The address of the end of the range (inclusive), or `null` to signify an empty range.

7.13.1 Detailed Description

Represents a range of characters in a MuPDFStructuredTextPage.

Definition at line 1254 of file MuPDFStructuredTextPage.cs.

7.13.2 Constructor & Destructor Documentation

7.13.2.1 MuPDFStructuredTextAddressSpan()

```
MuPDFCore.MuPDFStructuredTextAddressSpan.MuPDFStructuredTextAddressSpan (
            MuPDFStructuredTextAddress start,
            MuPDFStructuredTextAddress?  end )
```

Creates a new MuPDFStructuredTextAddressSpan corresponding to the specified character range.

Parameters

start	The addres of the start of the range.
end	The address of the end of the range (inclusive), or `null` to signify an empty range.

Definition at line 1271 of file MuPDFStructuredTextPage.cs.

7.13.3 Member Data Documentation

7.13.3.1 End

```
readonly?  MuPDFStructuredTextAddress MuPDFCore.MuPDFStructuredTextAddressSpan.End
```

The address of the end of the range (inclusive), or `null` to signify an empty range.

Definition at line 1264 of file MuPDFStructuredTextPage.cs.

7.13.3.2 Start

```
readonly MuPDFStructuredTextAddress MuPDFCore.MuPDFStructuredTextAddressSpan.Start
```

The addres of the start of the range.

Definition at line 1259 of file MuPDFStructuredTextPage.cs.

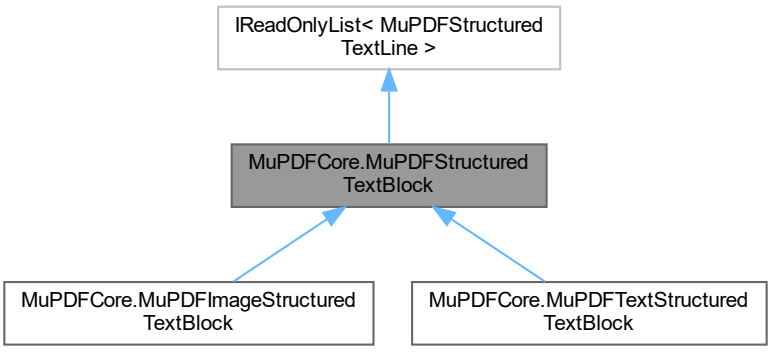
The documentation for this class was generated from the following file:

- MuPDFCore/MuPDFStructuredTextPage.cs

7.14 MuPDFCore.MuPDFStructuredTextBlock Class Reference

Represents a structured text block containing text or an image.

Inheritance diagram for MuPDFCore.MuPDFStructuredTextBlock:



Public Types

- enum Types

  Defines the type of the block.

Public Member Functions

- abstract IEnumerator< MuPDFStructuredTextLine > GetEnumerator ()

Properties

- abstract Types Type [get]

  The type of the block.
- Rectangle BoundingBox [get]

  The bounding box of the block.
- abstract int Count [get]

  The number of lines in the block.
- abstract MuPDFStructuredTextLine this[int index] [get]

  Gets the specified line from the block.

7.14.1 Detailed Description

Represents a structured text block containing text or an image.

Definition at line 556 of file MuPDFStructuredTextPage.cs.

7.14.2 Member Enumeration Documentation

7.14.2.1 Types

```
enum MuPDFCore.MuPDFStructuredTextBlock.Types
```

Defines the type of the block.

Definition at line 561 of file MuPDFStructuredTextPage.cs.

7.14.3 Member Function Documentation

7.14.3.1 GetEnumerator()

```
abstract IEnumerator< MuPDFStructuredTextLine > MuPDFCore.MuPDFStructuredTextBlock.GetEnumerator
( )  [pure virtual]
```

Implemented in MuPDFCore.MuPDFImageStructuredTextBlock, and MuPDFCore.MuPDFTextStructuredTextBlock.

7.14.4 Property Documentation

7.14.4.1 BoundingBox

```
Rectangle MuPDFCore.MuPDFStructuredTextBlock.BoundingBox  [get]
```

The bounding box of the block.

Definition at line 582 of file MuPDFStructuredTextPage.cs.

7.14.4.2 Count

```
abstract int MuPDFCore.MuPDFStructuredTextBlock.Count  [get]
```

The number of lines in the block.

Definition at line 587 of file MuPDFStructuredTextPage.cs.

7.14.4.3 this[int index]

```
abstract MuPDFStructuredTextLine MuPDFCore.MuPDFStructuredTextBlock.this[int index]  [get]
```

Gets the specified line from the block.

Parameters

index	The index of the line to extract.

Returns

The MuPDFStructuredTextLine with the specified *index* .

Definition at line 594 of file MuPDFStructuredTextPage.cs.

7.14.4.4 Type

```
abstract Types MuPDFCore.MuPDFStructuredTextBlock.Type  [get]
```

The type of the block.

Definition at line 577 of file MuPDFStructuredTextPage.cs.

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

- MuPDFCore/MuPDFStructuredTextPage.cs

7.15 MuPDFCore.MuPDFStructuredTextCharacter Class Reference

Represents a single text character.

Public Member Functions

- override string ToString ()

  Returns a string representation of the character.

Properties

- int CodePoint `[get]`

  The unicode code point of the character.
- string Character `[get]`

  A string representation of the character. It may consist of a single char or of a surrogate pair of chars.
- int Color `[get]`

  An sRGB hex representation of the colour of the character.
- PointF Origin `[get]`

  The baseline origin of the character.
- Quad BoundingQuad `[get]`

  A quadrilater bound for the character. This may or may not be a rectangle.
- float Size `[get]`

  The size in points of the character.

7.15.1 Detailed Description

Represents a single text character.

Definition at line 906 of file MuPDFStructuredTextPage.cs.

7.15.2 Member Function Documentation

7.15.2.1 ToString()

```
override string MuPDFCore.MuPDFStructuredTextCharacter.ToString ( )
```

Returns a string representation of the character.

Returns

A string representation of the character.

Definition at line 952 of file MuPDFStructuredTextPage.cs.

7.15.3 Property Documentation

7.15.3.1 BoundingQuad

```
Quad MuPDFCore.MuPDFStructuredTextCharacter.BoundingQuad [get]
```

A quadrilater bound for the character. This may or may not be a rectangle.

Definition at line 931 of file MuPDFStructuredTextPage.cs.

7.15.3.2 Character

```
string MuPDFCore.MuPDFStructuredTextCharacter.Character [get]
```

A string representation of the character. It may consist of a single char or of a surrogate pair of chars.

Definition at line 916 of file MuPDFStructuredTextPage.cs.

7.15.3.3 CodePoint

```
int MuPDFCore.MuPDFStructuredTextCharacter.CodePoint [get]
```

The unicode code point of the character.

Definition at line 911 of file MuPDFStructuredTextPage.cs.

7.15.3.4 Color

```
int MuPDFCore.MuPDFStructuredTextCharacter.Color [get]
```

An sRGB hex representation of the colour of the character.

Definition at line 921 of file MuPDFStructuredTextPage.cs.

7.15.3.5 Origin

```
PointF MuPDFCore.MuPDFStructuredTextCharacter.Origin [get]
```

The baseline origin of the character.

Definition at line 926 of file MuPDFStructuredTextPage.cs.

7.15.3.6 Size

```
float MuPDFCore.MuPDFStructuredTextCharacter.Size  [get]
```

The size in points of the character.

Definition at line 936 of file MuPDFStructuredTextPage.cs.

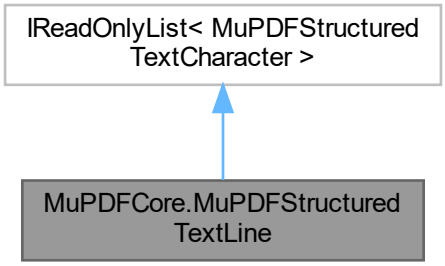
The documentation for this class was generated from the following file:

- MuPDFCore/MuPDFStructuredTextPage.cs

7.16 MuPDFCore.MuPDFStructuredTextLine Class Reference

Represents a single line of text (i.e. characters that share a common baseline).

Inheritance diagram for MuPDFCore.MuPDFStructuredTextLine:



Public Types

- enum WritingModes

  Defines the writing mode of the text.

Public Member Functions

- override string ToString ()

  Returns a string representation of the line.
- IEnumerator< MuPDFStructuredTextCharacter > GetEnumerator ()

Properties

- WritingModes WritingMode [get]

  The writing mode of the text.
- PointF Direction [get]

  The normalised direction of the text baseline.
- Rectangle BoundingBox [get]

  The bounding box of the line.
- MuPDFStructuredTextCharacter[ ] Characters [get]

  The characters contained in the line.
- string Text [get]

  A string representation of the characters contained in the line.
- int Count [get]

  The number of characters in the line.
- MuPDFStructuredTextCharacter this[int index] [get]

  Gets the specified character from the line.

7.16.1 Detailed Description

Represents a single line of text (i.e. characters that share a common baseline).

Definition at line 752 of file MuPDFStructuredTextPage.cs.

7.16.2 Member Enumeration Documentation

7.16.2.1 WritingModes

```
enum MuPDFCore.MuPDFStructuredTextLine.WritingModes
```

Defines the writing mode of the text.

Definition at line 757 of file MuPDFStructuredTextPage.cs.

7.16.3 Member Function Documentation

7.16.3.1 GetEnumerator()

```
IEnumerator< MuPDFStructuredTextCharacter > MuPDFCore.MuPDFStructuredTextLine.GetEnumerator (
)
```

Definition at line 892 of file MuPDFStructuredTextPage.cs.

7.16.3.2 ToString()

```
override string MuPDFCore.MuPDFStructuredTextLine.ToString ( )
```

Returns a string representation of the line.

Returns

A string representation of the line.

Definition at line 886 of file MuPDFStructuredTextPage.cs.

7.16.4 Property Documentation

7.16.4.1 BoundingBox

```
Rectangle MuPDFCore.MuPDFStructuredTextLine.BoundingBox  [get]
```

The bounding box of the line.

Definition at line 783 of file MuPDFStructuredTextPage.cs.

7.16.4.2 Characters

```
MuPDFStructuredTextCharacter [] MuPDFCore.MuPDFStructuredTextLine.Characters  [get]
```

The characters contained in the line.

Definition at line 788 of file MuPDFStructuredTextPage.cs.

7.16.4.3 Count

```
int MuPDFCore.MuPDFStructuredTextLine.Count  [get]
```

The number of characters in the line.

Definition at line 798 of file MuPDFStructuredTextPage.cs.

7.16.4.4 Direction

```
PointF MuPDFCore.MuPDFStructuredTextLine.Direction [get]
```

The normalised direction of the text baseline.

Definition at line 778 of file MuPDFStructuredTextPage.cs.

7.16.4.5 Text

```
string MuPDFCore.MuPDFStructuredTextLine.Text [get]
```

A string representation of the characters contained in the line.

Definition at line 793 of file MuPDFStructuredTextPage.cs.

7.16.4.6 this[int index]

```
MuPDFStructuredTextCharacter MuPDFCore.MuPDFStructuredTextLine.this[int index] [get]
```

Gets the specified character from the line.

Parameters

index	The index of the character.

Returns

The MuPDFStructuredTextCharacter with the specified *index* .

Definition at line 805 of file MuPDFStructuredTextPage.cs.

7.16.4.7 WritingMode

```
WritingModes MuPDFCore.MuPDFStructuredTextLine.WritingMode [get]
```

The writing mode of the text.

Definition at line 773 of file MuPDFStructuredTextPage.cs.

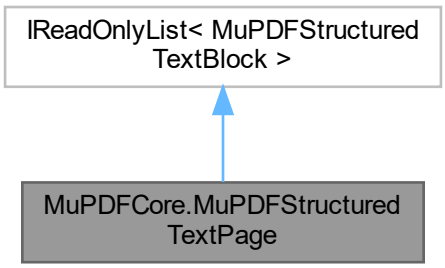
The documentation for this class was generated from the following file:

- MuPDFCore/MuPDFStructuredTextPage.cs

7.17 MuPDFCore.MuPDFStructuredTextPage Class Reference

Represents a structured representation of the text contained in a page.

Inheritance diagram for MuPDFCore.MuPDFStructuredTextPage:



Public Member Functions

- MuPDFStructuredTextAddress? GetHitAddress (PointF point, bool includeImages)

  Gets the address of the character that contains the specified point in page units.
- MuPDFStructuredTextAddress? GetClosestHitAddress (PointF point, bool includeImages)

  Gets the address of the character that contains the specified point in page units.
- IEnumerable< Quad > GetHighlightQuads (MuPDFStructuredTextAddressSpan range, bool includeImages)

  Gets a collection of Quads delimiting the specified character range . Where possible, these are collapsed at the line and block level. Each Quad may or may not be a rectangle.
- string GetText (MuPDFStructuredTextAddressSpan range)

  Gets the text corresponding to the specified character range . Blocks containing images are ignored.
- IEnumerable< MuPDFStructuredTextAddressSpan > Search (Regex needle)

  Searches for the specified Regex in the text of the page. A single match cannot span multiple lines.
- IEnumerator< MuPDFStructuredTextBlock > GetEnumerator ()

Properties

- MuPDFStructuredTextBlock[] StructuredTextBlocks `[get]`

  The blocks contained in the page.
- int Count `[get]`

  The number of blocks in the page.
- MuPDFStructuredTextBlock this[int index] `[get]`

  Gets the specified block in the page.
- MuPDFStructuredTextCharacter this[MuPDFStructuredTextAddress address] `[get]`

  Gets the specified character in the page.

7.17.1 Detailed Description

Represents a structured representation of the text contained in a page.

Definition at line 31 of file MuPDFStructuredTextPage.cs.

7.17.2 Member Function Documentation

7.17.2.1 GetClosestHitAddress()

```
MuPDFStructuredTextAddress?  MuPDFCore.MuPDFStructuredTextPage.GetClosestHitAddress (
            PointF point,
            bool includeImages )
```

Gets the address of the character that contains the specified *point* in page units.

Parameters

point	The point that must be closest to the character. This is expressed in page units (i.e. with a zoom factor of 1).
includeImages	If this is `true`, blocks containing images may be returned. Otherwise, only blocks containing text are considered.

Returns

The address of the character closest to the specified *point* This is `null` only if the page contains no characters.

Definition at line 208 of file MuPDFStructuredTextPage.cs.

7.17.2.2 GetEnumerator()

```
IEnumerator< MuPDFStructuredTextBlock > MuPDFCore.MuPDFStructuredTextPage.GetEnumerator ( )
```

Definition at line 542 of file MuPDFStructuredTextPage.cs.

7.17.2.3 GetHighlightQuads()

```
IEnumerable< Quad > MuPDFCore.MuPDFStructuredTextPage.GetHighlightQuads (
            MuPDFStructuredTextAddressSpan range,
            bool includeImages )
```

Gets a collection of Quads delimiting the specified character *range* . Where possible, these are collapsed at the line and block level. Each Quad may or may not be a rectangle.

Parameters

range	A MuPDFStructuredTextAddressSpan representing the character range
includeImages	If this is `true`, the bounding boxes for blocks containing images are also returned. Otherwise, only blocks containing text are considered.

Returns

A lazy collection of Quads delimiting the characters in the specified *includeImages* .

Definition at line 281 of file MuPDFStructuredTextPage.cs.

7.17.2.4 GetHitAddress()

```
MuPDFStructuredTextAddress?  MuPDFCore.MuPDFStructuredTextPage.GetHitAddress (
            PointF point,
            bool includeImages )
```

Gets the address of the character that contains the specified *point* in page units.

Parameters

point	The point that must be contained by the character. This is expressed in page units (i.e. with a zoom factor of 1).
includeImages	If this is `true`, blocks containing images may be returned. Otherwise, only blocks containing text are considered.

Returns

The address of the character containing the specified *point* , or `null` if no character contains the *point* .

Definition at line 174 of file MuPDFStructuredTextPage.cs.

7.17.2.5 GetText()

```
string MuPDFCore.MuPDFStructuredTextPage.GetText (
            MuPDFStructuredTextAddressSpan range )
```

Gets the text corresponding to the specified character *range* . Blocks containing images are ignored.

Parameters

range	A MuPDFStructuredTextAddressSpan representing the range of text to extract.

Returns

A string representation of the text contained in the specified *range* .

Definition at line 386 of file MuPDFStructuredTextPage.cs.

7.17.2.6 Search()

```
IEnumerable< MuPDFStructuredTextAddressSpan > MuPDFCore.MuPDFStructuredTextPage.Search (
            Regex needle )
```

Searches for the specified Regex in the text of the page. A single match cannot span multiple lines.

Parameters

needle	The Regex to search for.

Returns

A lazy collection of MuPDFStructuredTextAddressSpans representing all the occurrences of the *needle* in the text.

Definition at line 497 of file MuPDFStructuredTextPage.cs.

7.17.3 Property Documentation

7.17.3.1 Count

```
int MuPDFCore.MuPDFStructuredTextPage.Count  [get]
```

The number of blocks in the page.

Definition at line 41 of file MuPDFStructuredTextPage.cs.

7.17.3.2 StructuredTextBlocks

```
MuPDFStructuredTextBlock [] MuPDFCore.MuPDFStructuredTextPage.StructuredTextBlocks  [get]
```

The blocks contained in the page.

Definition at line 36 of file MuPDFStructuredTextPage.cs.

7.17.3.3 this[int index]

```
MuPDFStructuredTextBlock MuPDFCore.MuPDFStructuredTextPage.this[int index]  [get]
```

Gets the specified block in the page.

Parameters

index	The index of the block.

Returns

The block with the specified *index* .

Definition at line 48 of file MuPDFStructuredTextPage.cs.

7.17.3.4 this[MuPDFStructuredTextAddress address]

```
MuPDFStructuredTextCharacter MuPDFCore.MuPDFStructuredTextPage.this[MuPDFStructuredTextAddress
address]  [get]
```

Gets the specified character in the page.

Parameters

address	The address (block, line and character index) of the character.

Returns

A MuPDFStructuredTextCharacter representing the specified character.

Definition at line 55 of file MuPDFStructuredTextPage.cs.

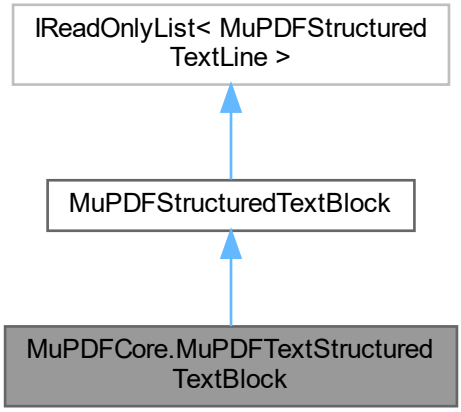
The documentation for this class was generated from the following file:

- MuPDFCore/MuPDFStructuredTextPage.cs

7.18 MuPDFCore.MuPDFTextStructuredTextBlock Class Reference

Represents a block containing multiple lines of text (typically a paragraph).

Inheritance diagram for MuPDFCore.MuPDFTextStructuredTextBlock:



Public Member Functions

- override IEnumerator< MuPDFStructuredTextLine > GetEnumerator ()
- override string ToString ()

  Returns the text contained in the block as a string.

Properties

- override Types Type `[get]`
- MuPDFStructuredTextLine[ ] Lines `[get]`

  The lines of text in the block.
- override int Count `[get]`
- override MuPDFStructuredTextLine this[int index] `[get]`

Additional Inherited Members

7.18.1 Detailed Description

Represents a block containing multiple lines of text (typically a paragraph).

Definition at line 656 of file MuPDFStructuredTextPage.cs.

7.18.2 Member Function Documentation

7.18.2.1 GetEnumerator()

```
override IEnumerator< MuPDFStructuredTextLine > MuPDFCore.MuPDFTextStructuredTextBlock.Get↩
Enumerator ( )  [virtual]
```

Implements MuPDFCore.MuPDFStructuredTextBlock.

Definition at line 727 of file MuPDFStructuredTextPage.cs.

7.18.2.2 ToString()

```
override string MuPDFCore.MuPDFTextStructuredTextBlock.ToString ( )
```

Returns the text contained in the block as a string.

Returns

The text contained in the block as a string. If the block contains at least one line, the return value has a line terminator at the end.

Definition at line 736 of file MuPDFStructuredTextPage.cs.

7.18.3 Property Documentation

7.18.3.1 Count

```
override int MuPDFCore.MuPDFTextStructuredTextBlock.Count  [get]
```

Definition at line 667 of file MuPDFStructuredTextPage.cs.

7.18.3.2 Lines

```
MuPDFStructuredTextLine [] MuPDFCore.MuPDFTextStructuredTextBlock.Lines  [get]
```

The lines of text in the block.

Definition at line 664 of file MuPDFStructuredTextPage.cs.

7.18.3.3 this[int index]

```
override MuPDFStructuredTextLine MuPDFCore.MuPDFTextStructuredTextBlock.this[int index]  [get]
```

Definition at line 670 of file MuPDFStructuredTextPage.cs.

7.18.3.4 Type

```
override Types MuPDFCore.MuPDFTextStructuredTextBlock.Type  [get]
```

Definition at line 659 of file MuPDFStructuredTextPage.cs.

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

- MuPDFCore/MuPDFStructuredTextPage.cs

7.19 MuPDFCore.OCRProgressInfo Class Reference

Describes OCR progress.

Properties

- double Progress [get]
  A value between 0 and 1, indicating how much progress has been completed.

7.19.1 Detailed Description

Describes OCR progress.

Definition at line 15 of file MuPDFStructuredTextPage.cs.

7.19.2 Property Documentation

7.19.2.1 Progress

```
double MuPDFCore.OCRProgressInfo.Progress  [get]
```

A value between 0 and 1, indicating how much progress has been completed.

Definition at line 20 of file MuPDFStructuredTextPage.cs.

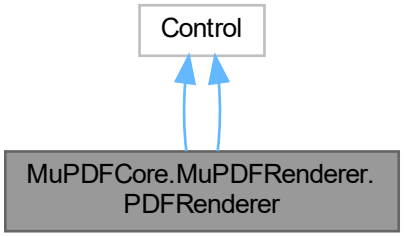
The documentation for this class was generated from the following file:

- MuPDFCore/MuPDFStructuredTextPage.cs

7.20 MuPDFCore.MuPDFRenderer.PDFRenderer Class Reference

A control to render PDF documents (and other formats), potentally using multiple threads.

Inheritance diagram for MuPDFCore.MuPDFRenderer.PDFRenderer:



Public Types

- enum PointerEventHandlers

  Identifies the action to perform on pointer events.

Public Member Functions

- PDFRenderer ()

  Initializes a new instance of the PDFRenderer class.
- void Initialize (MuPDFDocument document, int threadCount=0, int pageNumber=0, double resolution↩ Multiplier=1, bool includeAnnotations=true, TesseractLanguage ocrLanguage=null)

  Set up the PDFRenderer to display a page of a MuPDFDocument.
- async Task InitializeAsync (MuPDFDocument document, int threadCount=0, int pageNumber=0, double resolutionMultiplier=1, bool includeAnnotations=true, TesseractLanguage ocrLanguage=null, Cancellation↩ Token ocrCancellationToken=default, IProgress< OCRProgressInfo > ocrProgress=null)

  Set up the PDFRenderer to display a page of a MuPDFDocument. The OCR step is run asynchronously, in order not to block the UI thread.
- void Initialize (string fileName, int threadCount=0, int pageNumber=0, double resolutionMultiplier=1, bool includeAnnotations=true, TesseractLanguage ocrLanguage=null)

  Set up the PDFRenderer to display a page of a document that will be loaded from disk.
- async Task InitializeAsync (string fileName, int threadCount=0, int pageNumber=0, double resolution↩ Multiplier=1, bool includeAnnotations=true, TesseractLanguage ocrLanguage=null, CancellationToken ocr↩ CancellationToken=default, IProgress< OCRProgressInfo > ocrProgress=null)

  Set up the PDFRenderer to display a page of a document that will be loaded from disk. The OCR step is run asynchronously, in order not to block the UI thread.
- void Initialize (MemoryStream ms, InputFileTypes fileType, int threadCount=0, int pageNumber=0, double resolutionMultiplier=1, bool includeAnnotations=true, TesseractLanguage ocrLanguage=null)

  Set up the PDFRenderer to display a page of a document that will be loaded from a MemoryStream.
- async Task InitializeAsync (MemoryStream ms, InputFileTypes fileType, int threadCount=0, int page↩ Number=0, double resolutionMultiplier=1, bool includeAnnotations=true, TesseractLanguage ocr↩ Language=null, CancellationToken ocrCancellationToken=default, IProgress< OCRProgressInfo > ocr↩ Progress=null)

Set up the PDFRenderer to display a page of a document that will be loaded from a MemoryStream. The OCR step is run asynchronously, in order not to block the UI thread.

- void Initialize (byte[] dataBytes, InputFileTypes fileType, int offset=0, int length=-1, int threadCount=0, int pageNumber=0, double resolutionMultiplier=1, bool includeAnnotations=true, TesseractLanguage ocr↩ Language=null)

  Set up the PDFRenderer to display a page of a document that will be loaded from an array of bytes.
- async Task InitializeAsync (byte[] dataBytes, InputFileTypes fileType, int offset=0, int length=-1, int threadCount=0, int pageNumber=0, double resolutionMultiplier=1, bool includeAnnotations=true, TesseractLanguage ocrLanguage=null, CancellationToken ocrCancellationToken=default, IProgress< OCRProgressInfo > ocrProgress=null)

  Set up the PDFRenderer to display a page of a document that will be loaded from an array of bytes. The OCR step is run asynchronously, in order not to block the UI thread.
- void ReleaseResources ()

  Release resources held by this PDFRenderer. This is not an irreversible step: using one of the Initialize overloads after calling this method will restore functionality.
- void SetDisplayAreaNow (Rect value)

  Set the current display area to the specified value , skipping all transitions.
- void ZoomStep (double count, Point? center=null)

  Zoom around a point.
- void Contain ()

  Alter the display area so that the whole page fits on screen.
- void Cover ()

  Alter the display area so that the page covers the whole surface of the PDFRenderer (even though parts of the page may be outside it).
- RenderProgress GetProgress ()

  Get the current rendering progress.
- string GetSelectedText ()

  Get the currently selected text.
- void SelectAll ()

  Selects all the text in the document.
- int Search (Regex needle)

  Highlights all matches of the specified Regex in the text and returns the number of matches found. Matches cannot span multiple lines.
- override void Render (DrawingContext context)

  Draw the rendered document.

Static Public Attributes

- static readonly DirectProperty< PDFRenderer, int > RenderThreadCountProperty = AvaloniaProperty.↩ RegisterDirect<PDFRenderer, int>(nameof(RenderThreadCount), o => o.RenderThreadCount)

  Defines the RenderThreadCount property.
- static readonly DirectProperty< PDFRenderer, int > PageNumberProperty = AvaloniaProperty.Register↩ Direct<PDFRenderer, int>(nameof(PageNumber), o => o.PageNumber)

  Defines the PageNumber property.
- static readonly DirectProperty< PDFRenderer, bool > IsViewerInitializedProperty = AvaloniaProperty.↩ RegisterDirect<PDFRenderer, bool>(nameof(IsViewerInitialized), o => o.IsViewerInitialized)

  Defines the IsViewerInitialized property.
- static readonly DirectProperty< PDFRenderer, Rect > PageSizeProperty = AvaloniaProperty.Register↩ Direct<PDFRenderer, Rect>(nameof(PageSize), o => o.PageSize)

  Defines the PageSize property.
- static readonly StyledProperty< Rect > DisplayAreaProperty = AvaloniaProperty.Register<PDFRenderer, Rect>(nameof(DisplayArea))

Defines the DisplayArea property.

- static readonly StyledProperty< double > ZoomIncrementProperty = AvaloniaProperty.Register<PDFRenderer, double>(nameof(ZoomIncrement), Math.Pow(2, 1.0 / 3.0), defaultBindingMode: Avalonia.Data.BindingMode.TwoWay)

  Defines the ZoomIncrement property.
- static readonly StyledProperty< IBrush > BackgroundProperty = AvaloniaProperty.Register<PDFRenderer, IBrush>(nameof(Background))

  Defines the Background property.
- static readonly StyledProperty< IBrush > PageBackgroundProperty = AvaloniaProperty.Register<PDFRenderer, IBrush>(nameof(PageBackground))

  Defines the PageBackground property.
- static readonly DirectProperty< PDFRenderer, double > ZoomProperty = AvaloniaProperty.RegisterDirect<PDFRenderer, double>(nameof(Zoom), o => o.Zoom, (o, v) => o.Zoom = v, defaultBindingMode: Avalonia.Data.BindingMode.TwoWay)

  Defines the Zoom property.
- static readonly StyledProperty< PointerEventHandlers > PointerEventHandlerTypeProperty = AvaloniaProperty.Register<PDFRenderer, PointerEventHandlers>(nameof(PointerEventHandlersType), PointerEventHandlers.PanHighlight)

  Defines the PointerEventHandlersType property.
- static readonly StyledProperty< bool > ZoomEnabledProperty = AvaloniaProperty.Register<PDFRenderer, bool>(nameof(ZoomEnabled), true)

  Defines the ZoomEnabled property.
- static readonly StyledProperty< MuPDFStructuredTextAddressSpan > SelectionProperty = AvaloniaProperty.Register<PDFRenderer, MuPDFStructuredTextAddressSpan>(nameof(Selection), null)

  Defines the Selection property.
- static readonly StyledProperty< IBrush > SelectionBrushProperty = AvaloniaProperty.Register<PDFRenderer, IBrush>(nameof(SelectionBrush), new SolidColorBrush(Color.FromArgb(96, 86, 180, 233)))

  Defines the SelectionBrush property.
- static readonly StyledProperty< IEnumerable< MuPDFStructuredTextAddressSpan > > HighlightedRegionsProperty = AvaloniaProperty.Register<PDFRenderer, IEnumerable<MuPDFStructuredTextAddressSpan>>(nameof(HighlightedRegion null)

  Defines the HighlightedRegions property.
- static readonly StyledProperty< IBrush > HighlightBrushProperty = AvaloniaProperty.Register<PDFRenderer, IBrush>(nameof(HighlightBrush), new SolidColorBrush(Color.FromArgb(96, 230, 159, 0)))

  Defines the HighlightBrush property.

Properties

- int RenderThreadCount `[get]`

  Exposes the number of threads that the current instance is using to render the document. Read-only.
- int PageNumber `[get]`

  Exposes the number of the page that the current instance is rendering. Read-only.
- bool IsViewerInitialized `[get]`

  Whether the current instance has been initialised with a document to render or not. Read-only.
- Rect PageSize `[get]`

  Exposes the size of the page that is drawn by the current instance (in page units).
- Rect DisplayArea `[get, set]`

  The region of the page (in page units) that is currently displayed by the current instance. This always has the same aspect ratio of the bounds of this control. When this is set, the value is sanitised so that the smallest rectangle with the correct aspect ratio containing the requested value is chosen.
- double ZoomIncrement `[get, set]`

  Determines by how much the scale will be increased/decreased by the ZoomStep(double, Point?) method. Set this to a value smaller than 1 to invert the zoom in/out direction.

- IBrush Background [get, set]

  The background colour of the control.
- IBrush PageBackground [get, set]

  The background colour to use for the page drawn by the control.
- double Zoom [get, set]

  The current zoom level. Setting this will change the DisplayArea appropriately, zooming around the center of the DisplayArea.
- PointerEventHandlers PointerEventHandlersType [get, set]

  Whether the default handlers for pointer events (which are used for panning around the page) should be enabled. If this is false, you will have to implement your own way to pan around the document by changing the DisplayArea.
- bool ZoomEnabled [get, set]

  Whether the default handlers for pointer wheel events (which are used for zooming in/out) should be enabled. If this is false, you will have to implement your own way to zoom by changing the DisplayArea.
- MuPDFStructuredTextAddressSpan Selection [get, set]

  The start and end of the currently selected text.
- IBrush SelectionBrush [get, set]

  The colour used to highlight the Selection.
- IEnumerable< MuPDFStructuredTextAddressSpan > HighlightedRegions [get, set]

  A collection of highlighted regions, e.g. as a result of a text search.
- IBrush HighlightBrush [get, set]

  The colour used to highlight the HighlightedRegions.

7.20.1 Detailed Description

A control to render PDF documents (and other formats), potentally using multiple threads.

Definition at line 42 of file PDFRenderer.cs.

7.20.2 Member Enumeration Documentation

7.20.2.1 PointerEventHandlers

```
enum MuPDFCore.MuPDFRenderer.PDFRenderer.PointerEventHandlers
```

Identifies the action to perform on pointer events.

Definition at line 246 of file PDFRenderer.Properties.cs.

7.20.3 Constructor & Destructor Documentation

7.20.3.1 PDFRenderer()

```
MuPDFCore.MuPDFRenderer.PDFRenderer.PDFRenderer ( )
```

Initializes a new instance of the PDFRenderer class.

Definition at line 203 of file PDFRenderer.cs.

7.20.4 Member Function Documentation

7.20.4.1 Contain()

```
void MuPDFCore.MuPDFRenderer.PDFRenderer.Contain ( )
```

Alter the display area so that the whole page fits on screen.

Definition at line 700 of file PDFRenderer.cs.

7.20.4.2 Cover()

```
void MuPDFCore.MuPDFRenderer.PDFRenderer.Cover ( )
```

Alter the display area so that the page covers the whole surface of the PDFRenderer (even though parts of the page may be outside it).

Definition at line 709 of file PDFRenderer.cs.

7.20.4.3 GetProgress()

```
RenderProgress MuPDFCore.MuPDFRenderer.PDFRenderer.GetProgress ( )
```

Get the current rendering progress.

Returns

A RenderProgress object with information about the rendering progress of each thread.

Definition at line 730 of file PDFRenderer.cs.

7.20.4.4 GetSelectedText()

```
string MuPDFCore.MuPDFRenderer.PDFRenderer.GetSelectedText ( )
```

Get the currently selected text.

Returns

The currently selected text.

Definition at line 739 of file PDFRenderer.cs.

7.20.4.5 Initialize() [1/4]

```
void MuPDFCore.MuPDFRenderer.PDFRenderer.Initialize (
            byte[] dataBytes,
            InputFileTypes fileType,
            int offset = 0,
            int length = -1,
            int threadCount = 0,
            int pageNumber = 0,
            double resolutionMultiplier = 1,
            bool includeAnnotations = true,
            TesseractLanguage ocrLanguage = null )
```

Set up the PDFRenderer to display a page of a document that will be loaded from an array of bytes.

Parameters

dataBytes	The bytes of the document that should be opened. The array will be copied and can be safely discarded/altered after this method returns.
fileType	The format of the document.
offset	The offset in the byte array at which the document starts.
length	The length of the document in bytes. If this is < 0, the whole array is used.
threadCount	The number of threads to use in the rendering. If this is 0, an appropriate number of threads based on the number of processors in the computer will be used. Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7. If this is not the case, the biggest number smaller than *threadCount* that satisfies this condition is used.
pageNumber	The index of the page that should be rendered. The first page has index 0.
resolutionMultiplier	This value can be used to increase or decrease the resolution at which the static renderisation of the page will be produced. If *resolutionMultiplier* is 1, the resolution will match the size (in screen units) of the PDFRenderer.
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included in the rendering. Otherwise, only the page contents are included.
ocrLanguage	The language to use for optical character recognition (OCR). If this is null, no OCR is performed.

Definition at line 403 of file PDFRenderer.cs.

7.20.4.6 Initialize() [2/4]

```
void MuPDFCore.MuPDFRenderer.PDFRenderer.Initialize (
            MemoryStream ms,
            InputFileTypes fileType,
            int threadCount = 0,
            int pageNumber = 0,
            double resolutionMultiplier = 1,
            bool includeAnnotations = true,
            TesseractLanguage ocrLanguage = null )
```

Set up the PDFRenderer to display a page of a document that will be loaded from a MemoryStream.

Parameters

ms	The MemoryStream containing the document that should be opened. This can be safely disposed after this method returns.
fileType	The format of the document.
threadCount	The number of threads to use in the rendering. If this is 0, an appropriate number of threads based on the number of processors in the computer will be used. Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7. If this is not the case, the biggest number smaller than *threadCount* that satisfies this condition is used.
pageNumber	The index of the page that should be rendered. The first page has index 0.
resolutionMultiplier	This value can be used to increase or decrease the resolution at which the static renderisation of the page will be produced. If *resolutionMultiplier* is 1, the resolution will match the size (in screen units) of the PDFRenderer.
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included in the rendering. Otherwise, only the page contents are included.
ocrLanguage	The language to use for optical character recognition (OCR). If this is null, no OCR is performed.

Definition at line 359 of file PDFRenderer.cs.

7.20.4.7 Initialize() [3/4]

```
void MuPDFCore.MuPDFRenderer.PDFRenderer.Initialize (
            MuPDFDocument document,
            int threadCount = 0,
            int pageNumber = 0,
            double resolutionMultiplier = 1,
            bool includeAnnotations = true,
            TesseractLanguage ocrLanguage = null )
```

Set up the PDFRenderer to display a page of a MuPDFDocument.

Parameters

document	The MuPDFDocument to render.
threadCount	The number of threads to use in the rendering. If this is 0, an appropriate number of threads based on the number of processors in the computer will be used. Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7. If this is not the case, the biggest number smaller than *threadCount* that satisfies this condition is used.

Parameters

pageNumber	The index of the page that should be rendered. The first page has index 0.
resolutionMultiplier	This value can be used to increase or decrease the resolution at which the static renderisation of the page will be produced. If *resolutionMultiplier* is 1, the resolution will match the size (in screen units) of the PDFRenderer.
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included in the rendering. Otherwise, only the page contents are included.
ocrLanguage	The language to use for optical character recognition (OCR). If this is null, no OCR is performed.

Definition at line 258 of file PDFRenderer.cs.

7.20.4.8 Initialize() [4/4]

```
void MuPDFCore.MuPDFRenderer.PDFRenderer.Initialize (
            string fileName,
            int threadCount = 0,
            int pageNumber = 0,
            double resolutionMultiplier = 1,
            bool includeAnnotations = true,
            TesseractLanguage ocrLanguage = null )
```

Set up the PDFRenderer to display a page of a document that will be loaded from disk.

Parameters

fileName	The path to the document that should be opened.
threadCount	The number of threads to use in the rendering. If this is 0, an appropriate number of threads based on the number of processors in the computer will be used. Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7. If this is not the case, the biggest number smaller than *threadCount* that satisfies this condition is used.
pageNumber	The index of the page that should be rendered. The first page has index 0.
resolutionMultiplier	This value can be used to increase or decrease the resolution at which the static renderisation of the page will be produced. If *resolutionMultiplier* is 1, the resolution will match the size (in screen units) of the PDFRenderer.
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included in the rendering. Otherwise, only the page contents are included.
ocrLanguage	The language to use for optical character recognition (OCR). If this is null, no OCR is performed.

Definition at line 308 of file PDFRenderer.cs.

7.20.4.9 InitializeAsync() [1/4]

```
async Task MuPDFCore.MuPDFRenderer.PDFRenderer.InitializeAsync (
            byte[] dataBytes,
```

```
        InputFileTypes fileType,
        int offset = 0,
        int length = -1,
        int threadCount = 0,
        int pageNumber = 0,
        double resolutionMultiplier = 1,
        bool includeAnnotations = true,
        TesseractLanguage ocrLanguage = null,
        CancellationToken ocrCancellationToken = default,
        IProgress< OCRProgressInfo > ocrProgress = null )
```

Set up the PDFRenderer to display a page of a document that will be loaded from an array of bytes. The OCR step is run asynchronously, in order not to block the UI thread.

Parameters

dataBytes	The bytes of the document that should be opened. The array will be copied and can be safely discarded/altered after this method returns.
fileType	The format of the document.
offset	The offset in the byte array at which the document starts.
length	The length of the document in bytes. If this is < 0, the whole array is used.
threadCount	The number of threads to use in the rendering. If this is 0, an appropriate number of threads based on the number of processors in the computer will be used. Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7. If this is not the case, the biggest number smaller than *threadCount* that satisfies this condition is used.
pageNumber	The index of the page that should be rendered. The first page has index 0.
resolutionMultiplier	This value can be used to increase or decrease the resolution at which the static renderisation of the page will be produced. If *resolutionMultiplier* is 1, the resolution will match the size (in screen units) of the PDFRenderer.
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included in the rendering. Otherwise, only the page contents are included.
ocrLanguage	The language to use for optical character recognition (OCR). If this is null, no OCR is performed.
ocrCancellationToken	A CancellationToken used to cancel the OCR operation.
ocrProgress	An IProgress<OCRProgressInfo> used to report OCR progress.

Definition at line 446 of file PDFRenderer.cs.

7.20.4.10 InitializeAsync() [2/4]

```
async Task MuPDFCore.MuPDFRenderer.PDFRenderer.InitializeAsync (
            MemoryStream ms,
            InputFileTypes fileType,
            int threadCount = 0,
            int pageNumber = 0,
            double resolutionMultiplier = 1,
            bool includeAnnotations = true,
            TesseractLanguage ocrLanguage = null,
            CancellationToken ocrCancellationToken = default,
            IProgress< OCRProgressInfo > ocrProgress = null )
```

Set up the PDFRenderer to display a page of a document that will be loaded from a MemoryStream. The OCR step is run asynchronously, in order not to block the UI thread.

Parameters

ms	The MemoryStream containing the document that should be opened. This can be safely disposed after this method returns.
fileType	The format of the document.
threadCount	The number of threads to use in the rendering. If this is 0, an appropriate number of threads based on the number of processors in the computer will be used. Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7. If this is not the case, the biggest number smaller than *threadCount* that satisfies this condition is used.
pageNumber	The index of the page that should be rendered. The first page has index 0.
resolutionMultiplier	This value can be used to increase or decrease the resolution at which the static renderisation of the page will be produced. If *resolutionMultiplier* is 1, the resolution will match the size (in screen units) of the PDFRenderer.
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included in the rendering. Otherwise, only the page contents are included.
ocrLanguage	The language to use for optical character recognition (OCR). If this is null, no OCR is performed.
ocrCancellationToken	A CancellationToken used to cancel the OCR operation.
ocrProgress	An IProgress<OCRProgressInfo> used to report OCR progress.

Definition at line 381 of file PDFRenderer.cs.

7.20.4.11 InitializeAsync() [3/4]

```
async Task MuPDFCore.MuPDFRenderer.PDFRenderer.InitializeAsync (
            MuPDFDocument document,
            int threadCount = 0,
            int pageNumber = 0,
            double resolutionMultiplier = 1,
            bool includeAnnotations = true,
            TesseractLanguage ocrLanguage = null,
            CancellationToken ocrCancellationToken = default,
            IProgress< OCRProgressInfo > ocrProgress = null )
```

Set up the PDFRenderer to display a page of a MuPDFDocument. The OCR step is run asynchronously, in order not to block the UI thread.

Parameters

document	The MuPDFDocument to render.
threadCount	The number of threads to use in the rendering. If this is 0, an appropriate number of threads based on the number of processors in the computer will be used. Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7. If this is not the case, the biggest number smaller than *threadCount* that satisfies this condition is used.
pageNumber	The index of the page that should be rendered. The first page has index 0.
resolutionMultiplier	This value can be used to increase or decrease the resolution at which the static renderisation of the page will be produced. If *resolutionMultiplier* is 1, the resolution will match the size (in screen units) of the PDFRenderer.
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included in the rendering. Otherwise, only the page contents are included.
ocrLanguage	The language to use for optical character recognition (OCR). If this is null, no OCR is performed.
ocrCancellationToken	A CancellationToken used to cancel the OCR operation.
ocrProgress	An IProgress<OCRProgressInfo> used to report OCR progress.

Definition at line 284 of file PDFRenderer.cs.

7.20.4.12 InitializeAsync() [4/4]

```
async Task MuPDFCore.MuPDFRenderer.PDFRenderer.InitializeAsync (
            string fileName,
            int threadCount = 0,
            int pageNumber = 0,
            double resolutionMultiplier = 1,
            bool includeAnnotations = true,
            TesseractLanguage ocrLanguage = null,
            CancellationToken ocrCancellationToken = default,
            IProgress< OCRProgressInfo > ocrProgress = null )
```

Set up the PDFRenderer to display a page of a document that will be loaded from disk. The OCR step is run asynchronously, in order not to block the UI thread.

Parameters

fileName	The path to the document that should be opened.
threadCount	The number of threads to use in the rendering. If this is 0, an appropriate number of threads based on the number of processors in the computer will be used. Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7. If this is not the case, the biggest number smaller than *threadCount* that satisfies this condition is used.
pageNumber	The index of the page that should be rendered. The first page has index 0.
resolutionMultiplier	This value can be used to increase or decrease the resolution at which the static renderisation of the page will be produced. If *resolutionMultiplier* is 1, the resolution will match the size (in screen units) of the PDFRenderer.
includeAnnotations	If this is `true`, annotations (e.g. signatures) are included in the rendering. Otherwise, only the page contents are included.
ocrLanguage	The language to use for optical character recognition (OCR). If this is null, no OCR is performed.
ocrCancellationToken	A CancellationToken used to cancel the OCR operation.
ocrProgress	An IProgress<OCRProgressInfo> used to report OCR progress.

Definition at line 334 of file PDFRenderer.cs.

7.20.4.13 ReleaseResources()

```
void MuPDFCore.MuPDFRenderer.PDFRenderer.ReleaseResources ( )
```

Release resources held by this PDFRenderer. This is not an irreversible step: using one of the Initialize overloads after calling this method will restore functionality.

Definition at line 621 of file PDFRenderer.cs.

7.20.4.14 Render()

```
override void MuPDFCore.MuPDFRenderer.PDFRenderer.Render (
            DrawingContext context )
```

Draw the rendered document.

Parameters

context	The drawing context on which to draw.

Definition at line 1301 of file PDFRenderer.cs.

7.20.4.15 Search()

```
int MuPDFCore.MuPDFRenderer.PDFRenderer.Search (
            Regex needle )
```

Highlights all matches of the specified Regex in the text and returns the number of matches found. Matches cannot span multiple lines.

Parameters

needle	The Regex to search for.

Returns

The number of matches that have been found.

Definition at line 768 of file PDFRenderer.cs.

7.20.4.16 SelectAll()

```
void MuPDFCore.MuPDFRenderer.PDFRenderer.SelectAll ( )
```

Selects all the text in the document.

Definition at line 747 of file PDFRenderer.cs.

7.20.4.17 SetDisplayAreaNow()

```
void MuPDFCore.MuPDFRenderer.PDFRenderer.SetDisplayAreaNow (
            Rect value )
```

Set the current display area to the specified *value* , skipping all transitions.

Parameters

value	The new display area.

Definition at line 662 of file PDFRenderer.cs.

7.20.4.18 ZoomStep()

```
void MuPDFCore.MuPDFRenderer.PDFRenderer.ZoomStep (
            double count,
            Point?  center = null )
```

Zoom around a point.

Parameters

count	Number of steps to zoom. Positive values indicate a zoom in, negative values a zoom out.
center	The point around which to center the zoom operation. If this is null, the center of the control is used.

Definition at line 675 of file PDFRenderer.cs.

7.20.5 Member Data Documentation

7.20.5.1 BackgroundProperty

```
readonly StyledProperty<IBrush> MuPDFCore.MuPDFRenderer.PDFRenderer.BackgroundProperty =
AvaloniaProperty.Register<PDFRenderer, IBrush>(nameof(Background))  [static]
```

Defines the Background property.

Definition at line 182 of file PDFRenderer.Properties.cs.

7.20.5.2 DisplayAreaProperty

```
readonly StyledProperty<Rect> MuPDFCore.MuPDFRenderer.PDFRenderer.DisplayAreaProperty = Avalonia↩
Property.Register<PDFRenderer, Rect>(nameof(DisplayArea))  [static]
```

Defines the DisplayArea property.

Definition at line 128 of file PDFRenderer.Properties.cs.

7.20.5.3 HighlightBrushProperty

```
readonly StyledProperty<IBrush> MuPDFCore.MuPDFRenderer.PDFRenderer.HighlightBrushProperty
= AvaloniaProperty.Register<PDFRenderer, IBrush>(nameof(HighlightBrush), new SolidColor↩
Brush(Color.FromArgb(96, 230, 159, 0)))  [static]
```

Defines the HighlightBrush property.

Definition at line 337 of file PDFRenderer.Properties.cs.

7.20.5.4 HighlightedRegionsProperty

```
readonly StyledProperty<IEnumerable<MuPDFStructuredTextAddressSpan> > MuPDFCore.MuPDFRenderer.↩
PDFRenderer.HighlightedRegionsProperty = AvaloniaProperty.Register<PDFRenderer, IEnumerable<MuPDFStructuredTe
null)  [static]
```

Defines the HighlightedRegions property.

Definition at line 324 of file PDFRenderer.Properties.cs.

7.20.5.5 IsViewerInitializedProperty

```
readonly DirectProperty<PDFRenderer, bool> MuPDFCore.MuPDFRenderer.PDFRenderer.IsViewer↩
InitializedProperty = AvaloniaProperty.RegisterDirect<PDFRenderer, bool>(nameof(IsViewerInitialized),
o => o.IsViewerInitialized)  [static]
```

Defines the IsViewerInitialized property.

Definition at line 80 of file PDFRenderer.Properties.cs.

7.20.5.6 PageBackgroundProperty

```
readonly StyledProperty<IBrush> MuPDFCore.MuPDFRenderer.PDFRenderer.PageBackgroundProperty =
AvaloniaProperty.Register<PDFRenderer, IBrush>(nameof(PageBackground))  [static]
```

Defines the PageBackground property.

Definition at line 195 of file PDFRenderer.Properties.cs.

7.20.5.7 PageNumberProperty

```
readonly DirectProperty<PDFRenderer, int> MuPDFCore.MuPDFRenderer.PDFRenderer.PageNumber↩
Property = AvaloniaProperty.RegisterDirect<PDFRenderer, int>(nameof(PageNumber), o => o.↩
PageNumber)  [static]
```

Defines the PageNumber property.

Definition at line 56 of file PDFRenderer.Properties.cs.

7.20.5.8 PageSizeProperty

```
readonly DirectProperty<PDFRenderer, Rect> MuPDFCore.MuPDFRenderer.PDFRenderer.PageSize↩
Property = AvaloniaProperty.RegisterDirect<PDFRenderer, Rect>(nameof(PageSize), o => o.Page↩
Size)  [static]
```

Defines the PageSize property.

Definition at line 104 of file PDFRenderer.Properties.cs.

7.20.5.9 PointerEventHandlerTypeProperty

```
readonly StyledProperty<PointerEventHandlers> MuPDFCore.MuPDFRenderer.PDFRenderer.Pointer↩
EventHandlerTypeProperty = AvaloniaProperty.Register<PDFRenderer, PointerEventHandlers>(nameof(PointerEventHa
PointerEventHandlers.PanHighlight)  [static]
```

Defines the PointerEventHandlersType property.

Definition at line 272 of file PDFRenderer.Properties.cs.

7.20.5.10 RenderThreadCountProperty

```
readonly DirectProperty<PDFRenderer, int> MuPDFCore.MuPDFRenderer.PDFRenderer.RenderThread↩
CountProperty = AvaloniaProperty.RegisterDirect<PDFRenderer, int>(nameof(RenderThreadCount), o
=> o.RenderThreadCount)  [static]
```

Defines the RenderThreadCount property.

Definition at line 32 of file PDFRenderer.Properties.cs.

7.20.5.11 SelectionBrushProperty

```
readonly StyledProperty<IBrush> MuPDFCore.MuPDFRenderer.PDFRenderer.SelectionBrushProperty
= AvaloniaProperty.Register<PDFRenderer, IBrush>(nameof(SelectionBrush), new SolidColor↩
Brush(Color.FromArgb(96, 86, 180, 233)))  [static]
```

Defines the SelectionBrush property.

Definition at line 311 of file PDFRenderer.Properties.cs.

7.20.5.12 SelectionProperty

```
readonly StyledProperty<MuPDFStructuredTextAddressSpan> MuPDFCore.MuPDFRenderer.PDFRenderer.↩
SelectionProperty = AvaloniaProperty.Register<PDFRenderer, MuPDFStructuredTextAddressSpan>(nameof(Selection),
null)  [static]
```

Defines the Selection property.

Definition at line 298 of file PDFRenderer.Properties.cs.

7.20.5.13 ZoomEnabledProperty

```
readonly StyledProperty<bool> MuPDFCore.MuPDFRenderer.PDFRenderer.ZoomEnabledProperty = Avalonia↩
Property.Register<PDFRenderer, bool>(nameof(ZoomEnabled), true)  [static]
```

Defines the ZoomEnabled property.

Definition at line 285 of file PDFRenderer.Properties.cs.

7.20.5.14 ZoomIncrementProperty

```
readonly StyledProperty<double> MuPDFCore.MuPDFRenderer.PDFRenderer.ZoomIncrementProperty =
AvaloniaProperty.Register<PDFRenderer, double>(nameof(ZoomIncrement), Math.Pow(2, 1.0 / 3.0),
defaultBindingMode:  Avalonia.Data.BindingMode.TwoWay)  [static]
```

Defines the ZoomIncrement property.

Definition at line 160 of file PDFRenderer.Properties.cs.

7.20.5.15 ZoomProperty

```
readonly DirectProperty<PDFRenderer, double> MuPDFCore.MuPDFRenderer.PDFRenderer.ZoomProperty
= AvaloniaProperty.RegisterDirect<PDFRenderer, double>(nameof(Zoom), o => o.Zoom, (o, v) =>
o.Zoom = v, defaultBindingMode:  Avalonia.Data.BindingMode.TwoWay)  [static]
```

Defines the Zoom property.

Definition at line 208 of file PDFRenderer.Properties.cs.

7.20.6 Property Documentation

7.20.6.1 Background

```
IBrush MuPDFCore.MuPDFRenderer.PDFRenderer.Background  [get], [set]
```

The background colour of the control.

Definition at line 186 of file PDFRenderer.Properties.cs.

7.20.6.2 DisplayArea

```
Rect MuPDFCore.MuPDFRenderer.PDFRenderer.DisplayArea  [get], [set]
```

The region of the page (in page units) that is currently displayed by the current instance. This always has the same aspect ratio of the bounds of this control. When this is set, the value is sanitised so that the smallest rectangle with the correct aspect ratio containing the requested value is chosen.

Definition at line 133 of file PDFRenderer.Properties.cs.

7.20.6.3 HighlightBrush

```
IBrush MuPDFCore.MuPDFRenderer.PDFRenderer.HighlightBrush  [get], [set]
```

The colour used to highlight the HighlightedRegions.

Definition at line 341 of file PDFRenderer.Properties.cs.

7.20.6.4 HighlightedRegions

```
IEnumerable<MuPDFStructuredTextAddressSpan> MuPDFCore.MuPDFRenderer.PDFRenderer.Highlighted↩
Regions  [get], [set]
```

A collection of highlighted regions, e.g. as a result of a text search.

Definition at line 328 of file PDFRenderer.Properties.cs.

7.20.6.5 IsViewerInitialized

```
bool MuPDFCore.MuPDFRenderer.PDFRenderer.IsViewerInitialized  [get]
```

Whether the current instance has been initialised with a document to render or not. Read-only.

Definition at line 88 of file PDFRenderer.Properties.cs.

7.20.6.6 PageBackground

```
IBrush MuPDFCore.MuPDFRenderer.PDFRenderer.PageBackground  [get], [set]
```

The background colour to use for the page drawn by the control.

Definition at line 199 of file PDFRenderer.Properties.cs.

7.20.6.7 PageNumber

```
int MuPDFCore.MuPDFRenderer.PDFRenderer.PageNumber  [get]
```

Exposes the number of the page that the current instance is rendering. Read-only.

Definition at line 64 of file PDFRenderer.Properties.cs.

7.20.6.8 PageSize

```
Rect MuPDFCore.MuPDFRenderer.PDFRenderer.PageSize  [get]
```

Exposes the size of the page that is drawn by the current instance (in page units).

Definition at line 112 of file PDFRenderer.Properties.cs.

7.20.6.9 PointerEventHandlersType

```
PointerEventHandlers MuPDFCore.MuPDFRenderer.PDFRenderer.PointerEventHandlersType [get], [set]
```

Whether the default handlers for pointer events (which are used for panning around the page) should be enabled. If this is false, you will have to implement your own way to pan around the document by changing the DisplayArea.

Definition at line 276 of file PDFRenderer.Properties.cs.

7.20.6.10 RenderThreadCount

```
int MuPDFCore.MuPDFRenderer.PDFRenderer.RenderThreadCount [get]
```

Exposes the number of threads that the current instance is using to render the document. Read-only.

Definition at line 40 of file PDFRenderer.Properties.cs.

7.20.6.11 Selection

```
MuPDFStructuredTextAddressSpan MuPDFCore.MuPDFRenderer.PDFRenderer.Selection [get], [set]
```

The start and end of the currently selected text.

Definition at line 302 of file PDFRenderer.Properties.cs.

7.20.6.12 SelectionBrush

```
IBrush MuPDFCore.MuPDFRenderer.PDFRenderer.SelectionBrush [get], [set]
```

The colour used to highlight the Selection.

Definition at line 315 of file PDFRenderer.Properties.cs.

7.20.6.13 Zoom

```
double MuPDFCore.MuPDFRenderer.PDFRenderer.Zoom [get], [set]
```

The current zoom level. Setting this will change the DisplayArea appropriately, zooming around the center of the DisplayArea.

Definition at line 216 of file PDFRenderer.Properties.cs.

7.20.6.14 ZoomEnabled

```
bool MuPDFCore.MuPDFRenderer.PDFRenderer.ZoomEnabled  [get], [set]
```

Whether the default handlers for pointer wheel events (which are used for zooming in/out) should be enabled. If this is false, you will have to implement your own way to zoom by changing the DisplayArea.

Definition at line 289 of file PDFRenderer.Properties.cs.

7.20.6.15 ZoomIncrement

```
double MuPDFCore.MuPDFRenderer.PDFRenderer.ZoomIncrement  [get], [set]
```

Determines by how much the scale will be increased/decreased by the ZoomStep(double, Point?) method. Set this to a value smaller than 1 to invert the zoom in/out direction.

Definition at line 164 of file PDFRenderer.Properties.cs.

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

- MuPDFCore.MuPDFRenderer/PDFRenderer.cs
- MuPDFCore.MuPDFRenderer/PDFRenderer.Properties.cs

7.21 MuPDFCore.PointF Struct Reference

Represents a point.

Public Member Functions

- PointF (float x, float y)

  Create a new PointF from the specified coordinates.

Public Attributes

- float X

  The horizontal coordinate of the point.
- float Y

  The vertical coordinate of the point.

7.21.1 Detailed Description

Represents a point.

Definition at line 566 of file Rectangles.cs.

7.21.2 Constructor & Destructor Documentation

7.21.2.1 PointF()

```
MuPDFCore.PointF.PointF (
            float x,
            float y )
```

Create a new PointF from the specified coordinates.

Parameters

x	The horizontal coordinate of the point.
y	The vertical coordinate of the point.

Definition at line 583 of file Rectangles.cs.

7.21.3 Member Data Documentation

7.21.3.1 X

```
float MuPDFCore.PointF.X
```

The horizontal coordinate of the point.

Definition at line 571 of file Rectangles.cs.

7.21.3.2 Y

```
float MuPDFCore.PointF.Y
```

The vertical coordinate of the point.

Definition at line 576 of file Rectangles.cs.

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

- MuPDFCore/Rectangles.cs

7.22 MuPDFCore.Quad Struct Reference

Represents a quadrilater (not necessarily a rectangle).

Public Member Functions

- Quad (PointF lowerLeft, PointF upperLeft, PointF upperRight, PointF lowerRight)

  Creates a new Quad from the specified points.
- bool Contains (PointF point)

  Checks whether this Quad contains a PointF.

Public Attributes

- PointF LowerLeft

  The lower left point of the quadrilater.
- PointF UpperLeft

  The upper left point of the quadrilater.
- PointF UpperRight

  The upper right point of the quadrilater.
- PointF LowerRight

  The lower right point of the quadrilater.

7.22.1 Detailed Description

Represents a quadrilater (not necessarily a rectangle).

Definition at line 593 of file Rectangles.cs.

7.22.2 Constructor & Destructor Documentation

7.22.2.1 Quad()

```
MuPDFCore.Quad.Quad (
            PointF lowerLeft,
            PointF upperLeft,
            PointF upperRight,
            PointF lowerRight )
```

Creates a new Quad from the specified points.

Parameters

lowerLeft	The lower left point of the quadrilater.
upperLeft	The upper left point of the quadrilater.
upperRight	The upper right point of the quadrilater.
lowerRight	The lower right point of the quadrilater.

Definition at line 622 of file Rectangles.cs.

7.22.3 Member Function Documentation

7.22.3.1 Contains()

```
bool MuPDFCore.Quad.Contains (
            PointF point )
```

Checks whether this Quad contains a PointF.

Parameters

point	The PointF to check.

Returns

A boolean value indicating whether this Quad contains the *point* .

Definition at line 635 of file Rectangles.cs.

7.22.4 Member Data Documentation

7.22.4.1 LowerLeft

```
PointF MuPDFCore.Quad.LowerLeft
```

The lower left point of the quadrilater.

Definition at line 598 of file Rectangles.cs.

7.22.4.2 LowerRight

```
PointF MuPDFCore.Quad.LowerRight
```

The lower right point of the quadrilater.

Definition at line 613 of file Rectangles.cs.

7.22.4.3 UpperLeft

```
PointF MuPDFCore.Quad.UpperLeft
```

The upper left point of the quadrilater.

Definition at line 603 of file Rectangles.cs.

7.22.4.4 UpperRight

```
PointF MuPDFCore.Quad.UpperRight
```

The upper right point of the quadrilater.

Definition at line 608 of file Rectangles.cs.

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

- MuPDFCore/Rectangles.cs

7.23 MuPDFCore.Rectangle Struct Reference

Represents a rectangle.

Public Member Functions

- Rectangle (float x0, float y0, float x1, float y1)

  Create a new Rectangle from the specified coordinates.
- Rectangle (double x0, double y0, double x1, double y1)

  Create a new Rectangle from the specified coordinates.
- RoundedRectangle Round ()

  Round the rectangle's coordinates to the closest integers.
- RoundedRectangle Round (double zoom)

  Round the rectangle's coordinates to the closest integers, applying the specified zoom factor.
- Rectangle[ ] Split (int divisions)

  Split the rectangle into the specified number of Rectangles.
- Rectangle Intersect (Rectangle other)

  Compute the intersection between this Rectangle and another one.
- bool Contains (Rectangle other)

  Checks whether this Rectangle contains another Rectangle.
- bool Contains (PointF point)

  Checks whether this Rectangle contains a PointF.
- Quad ToQuad ()

  Converts the Rectangle to a Quad.

Public Attributes

- float X0

  The left coordinate of the rectangle.
- float Y0

  The top coordinate of the rectangle.
- float X1

  The right coordinate of the rectangle.
- float Y1

  The bottom coordinate of the rectangle.

Properties

- float Width [get]

  The width of the rectangle.
- float Height [get]

  The height of the rectangle.

7.23.1 Detailed Description

Represents a rectangle.

Definition at line 326 of file Rectangles.cs.

7.23.2 Constructor & Destructor Documentation

7.23.2.1 Rectangle() [1/2]

```
MuPDFCore.Rectangle.Rectangle (
            float x0,
            float y0,
            float x1,
            float y1 )
```

Create a new Rectangle from the specified coordinates.

Parameters

x0	The left coordinate of the rectangle.
y0	The top coordinate of the rectangle.
x1	The right coordinate of the rectangle.
y1	The bottom coordinate of the rectangle.

Definition at line 365 of file Rectangles.cs.

7.23.2.2 Rectangle() [2/2]

```
MuPDFCore.Rectangle.Rectangle (
            double x0,
            double y0,
            double x1,
            double y1 )
```

Create a new Rectangle from the specified coordinates.

Parameters

x0	The left coordinate of the rectangle.
y0	The top coordinate of the rectangle.
x1	The right coordinate of the rectangle.
y1	The bottom coordinate of the rectangle.

Definition at line 380 of file Rectangles.cs.

7.23.3 Member Function Documentation

7.23.3.1 Contains() [1/2]

```
bool MuPDFCore.Rectangle.Contains (
            PointF point )
```

Checks whether this Rectangle contains a PointF.

Parameters

point	The PointF to check.

Returns

A boolean value indicating whether this Rectangle contains the *point* .

Definition at line 476 of file Rectangles.cs.

7.23.3.2 Contains() [2/2]

```
bool MuPDFCore.Rectangle.Contains (
            Rectangle other )
```

Checks whether this Rectangle contains another Rectangle.

Parameters

other	The Rectangle to check.

Returns

A boolean value indicating whether this Rectangle contains the *other* Rectangle.

Definition at line 466 of file Rectangles.cs.

7.23.3.3 Intersect()

```
Rectangle MuPDFCore.Rectangle.Intersect (
            Rectangle other )
```

Compute the intersection between this Rectangle and another one.

Parameters

other	The other Rectangle to intersect with this instance.

Returns

The intersection between the two Rectangles.

Definition at line 443 of file Rectangles.cs.

7.23.3.4 Round() [1/2]

```
RoundedRectangle MuPDFCore.Rectangle.Round ( )
```

Round the rectangle's coordinates to the closest integers.

Returns

A RoundedRectangle with the rounded coordinates.

Definition at line 392 of file Rectangles.cs.

7.23.3.5 Round() [2/2]

```
RoundedRectangle MuPDFCore.Rectangle.Round (
            double zoom )
```

Round the rectangle's coordinates to the closest integers, applying the specified zoom factor.

Parameters

zoom	The zoom factor to apply.

Returns

A RoundedRectangle with the rounded coordinates.

Definition at line 407 of file Rectangles.cs.

7.23.3.6 Split()

```
Rectangle[] MuPDFCore.Rectangle.Split (
            int divisions )
```

Split the rectangle into the specified number of Rectangles.

Parameters

divisions	The number of rectangles in which the rectangle should be split. This must be factorisable using only powers of 2, 3, 5 or 7. Otherwise, the biggest number smaller than *divisions* that satisfies this condition is used.

Returns

An array of Rectangles that when positioned properly cover the same area as this object.

Definition at line 422 of file Rectangles.cs.

7.23.3.7 ToQuad()

```
Quad MuPDFCore.Rectangle.ToQuad ( )
```

Converts the Rectangle to a Quad.

Returns

A Quad corresponding to the current Rectangle.

Definition at line 485 of file Rectangles.cs.

7.23.4 Member Data Documentation

7.23.4.1 X0

```
float MuPDFCore.Rectangle.X0
```

The left coordinate of the rectangle.

Definition at line 331 of file Rectangles.cs.

7.23.4.2 X1

```
float MuPDFCore.Rectangle.X1
```

The right coordinate of the rectangle.

Definition at line 341 of file Rectangles.cs.

7.23.4.3 Y0

```
float MuPDFCore.Rectangle.Y0
```

The top coordinate of the rectangle.

Definition at line 336 of file Rectangles.cs.

7.23.4.4 Y1

```
float MuPDFCore.Rectangle.Y1
```

The bottom coordinate of the rectangle.

Definition at line 346 of file Rectangles.cs.

7.23.5 Property Documentation

7.23.5.1 Height

```
float MuPDFCore.Rectangle.Height  [get]
```

The height of the rectangle.

Definition at line 356 of file Rectangles.cs.

7.23.5.2 Width

```
float MuPDFCore.Rectangle.Width  [get]
```

The width of the rectangle.

Definition at line 351 of file Rectangles.cs.

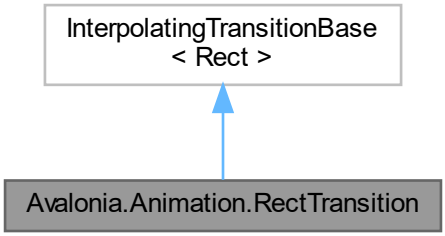
The documentation for this struct was generated from the following file:

- MuPDFCore/Rectangles.cs

7.24 Avalonia.Animation.RectTransition Class Reference

Transition class that handles AvaloniaProperty with Rect types.

Inheritance diagram for Avalonia.Animation.RectTransition:



7.24.1 Detailed Description

Transition class that handles AvaloniaProperty with Rect types.

Definition at line 23 of file RectTransition.cs.

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

- MuPDFCore.MuPDFRenderer/RectTransition.cs

7.25 MuPDFCore.RenderProgress Class Reference

Holds a summary of the progress of the current rendering operation.

Classes

- struct ThreadRenderProgress

  Holds the progress of a single thread.

Properties

- ThreadRenderProgress[] ThreadRenderProgresses [get]

  Contains the progress of all the threads used in rendering the document.

7.25.1 Detailed Description

Holds a summary of the progress of the current rendering operation.

Definition at line 383 of file MuPDF.cs.

7.25.2 Property Documentation

7.25.2.1 ThreadRenderProgresses

```
ThreadRenderProgress [] MuPDFCore.RenderProgress.ThreadRenderProgresses [get]
```

Contains the progress of all the threads used in rendering the document.

Definition at line 410 of file MuPDF.cs.

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

- MuPDFCore/MuPDF.cs

7.26 MuPDFCore.RoundedRectangle Struct Reference

Represents a rectangle using only integer numbers.

Public Member Functions

- RoundedRectangle (int x0, int y0, int x1, int y1)

  Create a new RoundedRectangle from the specified coordinates.
- RoundedRectangle[] Split (int divisions)

  Split the rectangle into the specified number of RoundedRectangles.

Public Attributes

- int X0

  The left coordinate of the rectangle.
- int Y0

  The top coordinate of the rectangle.
- int X1

  The right coordinate of the rectangle.
- int Y1

  The bottom coordinate of the rectangle.

Properties

- int Width `[get]`

  The width of the rectangle.
- int Height `[get]`

  The height of the rectangle.

7.26.1 Detailed Description

Represents a rectangle using only integer numbers.

Definition at line 494 of file Rectangles.cs.

7.26.2 Constructor & Destructor Documentation

7.26.2.1 RoundedRectangle()

```
MuPDFCore.RoundedRectangle.RoundedRectangle (
            int x0,
            int y0,
            int x1,
            int y1 )
```

Create a new RoundedRectangle from the specified coordinates.

Parameters

x0	The left coordinate of the rectangle.
y0	The top coordinate of the rectangle.
x1	The right coordinate of the rectangle.
y1	The bottom coordinate of the rectangle.

Definition at line 533 of file Rectangles.cs.

7.26.3 Member Function Documentation

7.26.3.1 Split()

```
RoundedRectangle[] MuPDFCore.RoundedRectangle.Split (
            int divisions )
```

Split the rectangle into the specified number of RoundedRectangles.

Parameters

divisions	The number of rectangles in which the rectangle should be split. This must be factorisable using only powers of 2, 3, 5 or 7. Otherwise, the biggest number smaller than *divisions* that satisfies this condition is used.

Returns

An array of RoundedRectangles that when positioned properly cover the same area as this object.

Definition at line 546 of file Rectangles.cs.

7.26.4 Member Data Documentation

7.26.4.1 X0

```
int MuPDFCore.RoundedRectangle.X0
```

The left coordinate of the rectangle.

Definition at line 499 of file Rectangles.cs.

7.26.4.2 X1

```
int MuPDFCore.RoundedRectangle.X1
```

The right coordinate of the rectangle.

Definition at line 509 of file Rectangles.cs.

7.26.4.3 Y0

```
int MuPDFCore.RoundedRectangle.Y0
```

The top coordinate of the rectangle.

Definition at line 504 of file Rectangles.cs.

7.26.4.4 Y1

```
int MuPDFCore.RoundedRectangle.Y1
```

The bottom coordinate of the rectangle.

Definition at line 514 of file Rectangles.cs.

7.26.5 Property Documentation

7.26.5.1 Height

```
int MuPDFCore.RoundedRectangle.Height  [get]
```

The height of the rectangle.

Definition at line 524 of file Rectangles.cs.

7.26.5.2 Width

```
int MuPDFCore.RoundedRectangle.Width  [get]
```

The width of the rectangle.

Definition at line 519 of file Rectangles.cs.

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

- MuPDFCore/Rectangles.cs

7.27 MuPDFCore.RoundedSize Struct Reference

Represents the size of a rectangle using only integer numbers.

Public Member Functions

- RoundedSize (int width, int height)

  Create a new RoundedSize with the specified width and height.
- RoundedRectangle[ ] Split (int divisions)

  Split the size into the specified number of RoundedRectangles.

Public Attributes

- int Width

  The width of the rectangle.
- int Height

  The height of the rectangle.

7.27.1 Detailed Description

Represents the size of a rectangle using only integer numbers.

Definition at line 181 of file Rectangles.cs.

7.27.2 Constructor & Destructor Documentation

7.27.2.1 RoundedSize()

```
MuPDFCore.RoundedSize.RoundedSize (
            int width,
            int height )
```

Create a new RoundedSize with the specified width and height.

Parameters

width	The width of the rectangle.
height	The height of the rectangle.

Definition at line 198 of file Rectangles.cs.

7.27.3 Member Function Documentation

7.27.3.1 Split()

```
RoundedRectangle[] MuPDFCore.RoundedSize.Split (
            int divisions )
```

Split the size into the specified number of RoundedRectangles.

Parameters

divisions	The number of rectangles in which the size should be split. This must be factorisable using only powers of 2, 3, 5 or 7. Otherwise, the biggest number smaller than *divisions* that satisfies this condition is used.

Returns

An array of RoundedRectangles that when positioned properly cover an area of the size of this object.

Definition at line 209 of file Rectangles.cs.

7.27.4 Member Data Documentation

7.27.4.1 Height

```
int MuPDFCore.RoundedSize.Height
```

The height of the rectangle.

Definition at line 191 of file Rectangles.cs.

7.27.4.2 Width

```
int MuPDFCore.RoundedSize.Width
```

The width of the rectangle.

Definition at line 186 of file Rectangles.cs.

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

- MuPDFCore/Rectangles.cs

7.28 MuPDFCore.Size Struct Reference

Represents the size of a rectangle.

Public Member Functions

- Size (float width, float height)

  Create a new Size with the specified width and height.
- Size (double width, double height)

  Create a new Size with the specified width and height.
- Rectangle[ ] Split (int divisions)

  Split the size into the specified number of Rectangles.

Public Attributes

- float Width

  The width of the rectangle.
- float Height

  The height of the rectangle.

7.28.1 Detailed Description

Represents the size of a rectangle.

Definition at line 25 of file Rectangles.cs.

7.28.2 Constructor & Destructor Documentation

7.28.2.1 Size() [1/2]

```
MuPDFCore.Size.Size (
            float width,
            float height )
```

Create a new Size with the specified width and height.

Parameters

width	The width of the rectangle.
height	The height of the rectangle.

Definition at line 42 of file Rectangles.cs.

7.28.2.2 Size() [2/2]

```
MuPDFCore.Size.Size (
            double width,
            double height )
```

Create a new Size with the specified width and height.

Parameters

width	The width of the rectangle.
height	The height of the rectangle.

Definition at line 53 of file Rectangles.cs.

7.28.3 Member Function Documentation

7.28.3.1 Split()

```
Rectangle[] MuPDFCore.Size.Split (
            int divisions )
```

Split the size into the specified number of Rectangles.

Parameters

divisions	The number of rectangles in which the size should be split. This must be factorisable using only powers of 2, 3, 5 or 7. Otherwise, the biggest number smaller than *divisions* that satisfies this condition is used.

Returns

An array of Rectangles that when positioned properly cover an area of the size of this object.

Definition at line 64 of file Rectangles.cs.

7.28.4 Member Data Documentation

7.28.4.1 Height

```
float MuPDFCore.Size.Height
```

The height of the rectangle.

Definition at line 35 of file Rectangles.cs.

7.28.4.2 Width

```
float MuPDFCore.Size.Width
```

The width of the rectangle.

Definition at line 30 of file Rectangles.cs.

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

- MuPDFCore/Rectangles.cs

7.29 MuPDFCore.TesseractLanguage Class Reference

Represents a language used by Tesseract OCR.

Public Types

- enum Fast

  Fast integer versions of trained models. These are models for a single language.
- enum FastScripts

  Fast integer versions of trained models. These are models for a single script supporting one or more languages.
- enum Best

  Best (most accurate) trained models. These are models for a single language.
- enum BestScripts

  Best (most accurate) trained models. These are models for a single script supporting one or more languages.

Public Member Functions

- TesseractLanguage (string prefix, string language)

  Create a new TesseractLanguage object using the provided prefix and language name, without processing them in any way.
- TesseractLanguage (string fileName)

  Create a new TesseractLanguage object using the specified trained model data file.
- TesseractLanguage (Fast language, bool useAnyCached=false)

  Create a new TesseractLanguage object using a fast integer version of a trained model for the specified language. The language file is downloaded from the `tesseract-ocr/tessdata_fast` GitHub repository. If it has already been downloaded and cached before, the downloaded file is re-used.
- TesseractLanguage (Best language, bool useAnyCached=false)

  Create a new TesseractLanguage object using the best (most accurate) version of the trained model for the specified language. The language file is downloaded from the `tesseract-ocr/tessdata_best` GitHub repository. If it has already been downloaded and cached before, the downloaded file is re-used.
- TesseractLanguage (FastScripts script, bool useAnyCached=false)

  Create a new TesseractLanguage object using a fast integer version of a trained model for the specified script. The language file is downloaded from the `tesseract-ocr/tessdata_fast` GitHub repository. If it has already been downloaded and cached before, the downloaded file is re-used.
- TesseractLanguage (BestScripts script, bool useAnyCached=false)

  Create a new TesseractLanguage object using the best (most accurate) version of the trained model for the specified script. The language file is downloaded from the `tesseract-ocr/tessdata_best` GitHub repository. If it has already been downloaded and cached before, the downloaded file is re-used.

Properties

- string Prefix [get]

  The name of the folder where the language file is located.
- string Language [get]

  The name of the language. The Tesseract library will assume that the trained language data file can be found at `Prefix/Language.traineddata`.

7.29.1 Detailed Description

Represents a language used by Tesseract OCR.

Definition at line 13 of file TesseractLanguage.cs.

7.29.2 Member Enumeration Documentation

7.29.2.1 Best

`enum MuPDFCore.TesseractLanguage.Best`

Best (most accurate) trained models. These are models for a single language.

Definition at line 690 of file TesseractLanguage.cs.

7.29.2.2 BestScripts

`enum MuPDFCore.TesseractLanguage.BestScripts`

Best (most accurate) trained models. These are models for a single script supporting one or more languages.

Definition at line 1193 of file TesseractLanguage.cs.

7.29.2.3 Fast

`enum MuPDFCore.TesseractLanguage.Fast`

Fast integer versions of trained models. These are models for a single language.

Definition at line 28 of file TesseractLanguage.cs.

7.29.2.4 FastScripts

```
enum MuPDFCore.TesseractLanguage.FastScripts
```

Fast integer versions of trained models. These are models for a single script supporting one or more languages.

Definition at line 535 of file TesseractLanguage.cs.

7.29.3 Constructor & Destructor Documentation

7.29.3.1 TesseractLanguage() [1/6]

```
MuPDFCore.TesseractLanguage.TesseractLanguage (
            string prefix,
            string language )
```

Create a new TesseractLanguage object using the provided *prefix* and *language* name, without processing them in any way.

Parameters

prefix	The name of the folder where the language file is located. If this is `null`, the value of the environment variable `TESSDATA_PREFIX` will be used.
language	The name of the language. The Tesseract library will assume that the trained language data file can be found at *prefix* / *language* `.traineddata`.

Definition at line 1350 of file TesseractLanguage.cs.

7.29.3.2 TesseractLanguage() [2/6]

```
MuPDFCore.TesseractLanguage.TesseractLanguage (
            string fileName )
```

Create a new TesseractLanguage object using the specified trained model data file.

Parameters

fileName	The path to the trained model data file. If the file name does not end in `.traineddata`, the file is copied to a temporary folder, and the temporary file is used by the Tesseract library.

Definition at line 1360 of file TesseractLanguage.cs.

7.29.3.3 TesseractLanguage() [3/6]

```
MuPDFCore.TesseractLanguage.TesseractLanguage (
            Fast language,
            bool useAnyCached = false )
```

Create a new TesseractLanguage object using a fast integer version of a trained model for the specified language. The language file is downloaded from the `tesseract-ocr/tessdata_fast` GitHub repository. If it has already been downloaded and cached before, the downloaded file is re-used.

Parameters

language	The language to use for the OCR process.
useAnyCached	If this is `true`, if a cached trained model file is available for the specified language, it will be used even if it is a "best (most accurate)" model. Otherwise, only cached fast integer trained models will be used.

Definition at line 1387 of file TesseractLanguage.cs.

7.29.3.4 TesseractLanguage() [4/6]

```
MuPDFCore.TesseractLanguage.TesseractLanguage (
            Best language,
            bool useAnyCached = false )
```

Create a new TesseractLanguage object using the best (most accurate) version of the trained model for the specified language. The language file is downloaded from the `tesseract-ocr/tessdata_best` GitHub repository. If it has already been downloaded and cached before, the downloaded file is re-used.

Parameters

language	The language to use for the OCR process.
useAnyCached	If this is `true`, if a cached trained model file is available for the specified language, it will be used even if it is a "fast" model. Otherwise, only cached best (most accurate) trained models will be used.

Definition at line 1453 of file TesseractLanguage.cs.

7.29.3.5 TesseractLanguage() [5/6]

```
MuPDFCore.TesseractLanguage.TesseractLanguage (
            FastScripts script,
            bool useAnyCached = false )
```

Create a new TesseractLanguage object using a fast integer version of a trained model for the specified script. The language file is downloaded from the `tesseract-ocr/tessdata_fast` GitHub repository. If it has already been downloaded and cached before, the downloaded file is re-used.

Parameters

script	The script to use for the OCR process.
useAnyCached	If this is `true`, if a cached trained model file is available for the specified script, it will be used even if it is a "best (most accurate)" model. Otherwise, only cached fast integer trained models will be used.

Definition at line 1519 of file TesseractLanguage.cs.

7.29.3.6 TesseractLanguage() [6/6]

```
MuPDFCore.TesseractLanguage.TesseractLanguage (
            BestScripts script,
            bool useAnyCached = false )
```

Create a new TesseractLanguage object using the best (most accurate) version of the trained model for the specified script. The language file is downloaded from the `tesseract-ocr/tessdata_best` GitHub repository. If it has already been downloaded and cached before, the downloaded file is re-used.

Parameters

script	The script to use for the OCR process.
useAnyCached	If this is `true`, if a cached trained model file is available for the specified script, it will be used even if it is a "fast" model. Otherwise, only cached best (most accurate) trained models will be used.

Definition at line 1589 of file TesseractLanguage.cs.

7.29.4 Property Documentation

7.29.4.1 Language

```
string MuPDFCore.TesseractLanguage.Language  [get]
```

The name of the language. The Tesseract library will assume that the trained language data file can be found at `Prefix/Language.traineddata`.

Definition at line 23 of file TesseractLanguage.cs.

7.29.4.2 Prefix

```
string MuPDFCore.TesseractLanguage.Prefix  [get]
```

The name of the folder where the language file is located.

Definition at line 18 of file TesseractLanguage.cs.

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

- MuPDFCore/TesseractLanguage.cs

7.30 MuPDFCore.RenderProgress.ThreadRenderProgress Struct Reference

Holds the progress of a single thread.

Public Attributes

- int Progress
  The current progress.
- long MaxProgress
  The maximum progress. If this is 0, this value could not be determined (yet).

7.30.1 Detailed Description

Holds the progress of a single thread.

Definition at line 388 of file MuPDF.cs.

7.30.2 Member Data Documentation

7.30.2.1 MaxProgress

```
long MuPDFCore.RenderProgress.ThreadRenderProgress.MaxProgress
```

The maximum progress. If this is 0, this value could not be determined (yet).

Definition at line 398 of file MuPDF.cs.

7.30.2.2 Progress

```
int MuPDFCore.RenderProgress.ThreadRenderProgress.Progress
```

The current progress.

Definition at line 393 of file MuPDF.cs.

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

- MuPDFCore/MuPDF.cs

Chapter 8

File Documentation

8.1 PDFRenderer.cs

```
00001 /*
00002 MuPDFCore.MuPDFRenderer - A control to display documents in Avalonia using MuPDFCore.
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 Avalonia;
00019 using Avalonia.Animation;
00020 using Avalonia.Controls;
00021 using Avalonia.Input;
00022 using Avalonia.Layout;
00023 using Avalonia.LogicalTree;
00024 using Avalonia.Media;
00025 using Avalonia.Media.Imaging;
00026 using Avalonia.Platform;
00027 using Avalonia.Threading;
00028 using System;
00029 using System.Collections.Generic;
00030 using System.IO;
00031 using System.Linq;
00032 using System.Runtime.InteropServices;
00033 using System.Text.RegularExpressions;
00034 using System.Threading;
00035 using System.Threading.Tasks;
00036
00037 namespace MuPDFCore.MuPDFRenderer
00038 {
00039 /// <summary>
00040 /// A control to render PDF documents (and other formats), potentally using multiple threads.
00041 /// </summary>
00042     public partial class PDFRenderer :  Control
00043     {
00044 /// <summary>
00045 /// If this is true, the <see cref="Context"/> and <see cref="Document"/> will be disposed when this
      object is detached from the logical tree.
00046 /// </summary>
00047         private bool OwnsContextAndDocument = true;
00048
00049 /// <summary>
00050 /// The <see cref="MuPDFContext"/> using which the <see cref="Document"/> was created.
00051 /// </summary>
00052         protected MuPDFContext Context;
00053
00054 /// <summary>
00055 /// The <see cref="MuPDFDocument"/> from which the <see cref="Renderer"/> was created.
00056 /// </summary>
00057         protected MuPDFDocument Document;
```

```
00058
00059 /// <summary>
00060 /// The <see cref="MuPDFMultiThreadedPageRenderer"/> that renders the dynamic tiles.
00061 /// </summary>
00062         private MuPDFMultiThreadedPageRenderer Renderer;
00063
00064 /// <summary>
00065 /// The static renderisation of the page.
00066 /// </summary>
00067         private WriteableBitmap FixedCanvasBitmap;
00068
00069 /// <summary>
00070 /// The area covered by the <see cref="FixedCanvasBitmap"/>.  It should be equal to the <see
      cref="PageSize"/>, but doesn't have to.
00071 /// </summary>
00072         private Rectangle FixedArea;
00073
00074 /// <summary>
00075 /// The position and size of the dynamic tiles.
00076 /// </summary>
00077         private RoundedRectangle[] DynamicImagesBounds;
00078
00079 /// <summary>
00080 /// The dynamic tiles.
00081 /// </summary>
00082         private WriteableBitmap[] DynamicBitmaps;
00083
00084 /// <summary>
00085 /// If this is true, the <see cref="DynamicBitmaps"/> have been rendered and can be drawn on screen.
00086 /// </summary>
00087         private bool AreDynamicBitmapsReady = false;
00088
00089 /// <summary>
00090 /// If this is true, the <see cref="DynamicBitmaps"/> will be rendered again immediately after the
      current rendering operation finishes.
00091 /// </summary>
00092         private bool RenderQueued = false;
00093
00094 /// <summary>
00095 /// A <see cref="Mutex"/> to synchronise rendering operations.  If someone else is holding this mutex,
      you can assume that it's not safe to access the <see cref="DynamicBitmaps"/>.
00096 /// </summary>
00097         private Mutex RenderMutex;
00098
00099 /// <summary>
00100 /// A <see cref="Geometry"/> holding the icon that is displayed in the top-right corner when the <see
      cref="DynamicBitmaps"/> are not available.
00101 /// </summary>
00102         private PathGeometry RefreshingGeometry;
00103
00104 /// <summary>
00105 /// The thread that is in charge of responding to the rendering requests and either starting a new
      rendering of the <see cref="DynamicBitmaps"/>, or queueing it.
00106 /// </summary>
00107         private Thread RenderDynamicCanvasOuterThread;
00108
00109 /// <summary>
00110 /// An <see cref="EventWaitHandle"/> that signals a request for rendering to the <see
      cref="RenderDynamicCanvasOuterThread"/>.
00111 /// </summary>
00112         private readonly EventWaitHandle RenderDynamicCanvasOuterHandle = new EventWaitHandle(false,
      EventResetMode.ManualReset);
00113
00114 /// <summary>
00115 /// The thread that is in charge of rendering the <see cref="DynamicBitmaps"/>.
00116 /// </summary>
00117         private Thread RenderDynamicCanvasInnerThread;
00118
00119 /// <summary>
00120 /// An <see cref="EventWaitHandle"/> that signals a request for rendering to the <see
      cref="RenderDynamicCanvasInnerThread"/>.
00121 /// </summary>
00122         private readonly EventWaitHandle RenderDynamicCanvasInnerHandle = new EventWaitHandle(false,
      EventResetMode.ManualReset);
00123
00124 /// <summary>
00125 /// An <see cref="EventWaitHandle"/> that signals to the <see cref="RenderDynamicCanvasOuterThread"/>
      that the <see cref="RenderDynamicCanvasInnerThread"/> has acquired the <see cref="RenderMutex"/> and
      is starting rendering.
00126 /// </summary>
00127         private readonly EventWaitHandle RenderDynamicCanvasInnerStartedHandle = new
      EventWaitHandle(false, EventResetMode.ManualReset);
00128
00129 /// <summary>
00130 /// An <see cref="EventWaitHandle"/> that signals to the <see cref="RenderDynamicCanvasOuterThread"/>
      and the <see cref="RenderDynamicCanvasInnerThread"/> to cease all operation because this <see
      cref="PDFRenderer"/> is being detached from the logical tree.
```

```
00131 /// </summary>
00132         private readonly EventWaitHandle RendererDisposedHandle = new EventWaitHandle(false,
      EventResetMode.ManualReset);
00133
00134 /// <summary>
00135 /// The current rendering resolution (in screen units) that is used by the renderer when rendering the
      <see cref="DynamicBitmaps"/>.
00136 /// </summary>
00137         private readonly int[] RenderSize = new int[2];
00138
00139 /// <summary>
00140 /// The area on the page that will be rendered by the renderer in the <see cref="DynamicBitmaps"/>.
00141 /// </summary>
00142         private Rect RenderDisplayArea;
00143
00144 /// <summary>
00145 /// A lock to prevent race conditions when multiple rendering passes are queued consecutively.
00146 /// </summary>
00147         private readonly object RenderDisplayAreaLock = new object();
00148
00149 /// <summary>
00150 /// Whether a PointerPressed event has fired.
00151 /// </summary>
00152         private bool IsMouseDown = false;
00153
00154 /// <summary>
00155 /// The point at which the PointerPressed event fired.
00156 /// </summary>
00157         private Point MouseDownPoint;
00158
00159 /// <summary>
00160 /// The <see cref="DisplayArea"/> when the PointerPressed event fired.
00161 /// </summary>
00162         private Rect MouseDownDisplayArea;
00163
00164 /// <summary>
00165 /// A structured text representation of the current page, used for selection and search highlight.
00166 /// </summary>
00167         protected MuPDFStructuredTextPage StructuredTextPage;
00168
00169 /// <summary>
00170 /// A list of <see cref="Quad"/>s that cover the selected text region.
00171 /// </summary>
00172         protected List<Quad> SelectionQuads;
00173
00174 /// <summary>
00175 /// A list of <see cref="Quad"/>s that cover the highlighted regions.
00176 /// </summary>
00177         protected List<Quad> HighlightQuads;
00178
00179 /// <summary>
00180 /// Defines the current mouse operation.
00181 /// </summary>
00182         private enum CurrentMouseOperations
00183         {
00184 /// <summary>
00185 /// The mouse is being used to pan around the page.
00186 /// </summary>
00187             Pan,
00188
00189 /// <summary>
00190 /// The mouse is being used to highlight text
00191 /// </summary>
00192             Highlight
00193         }
00194
00195 /// <summary>
00196 /// The current mouse operation.
00197 /// </summary>
00198         private CurrentMouseOperations CurrentMouseOperation;
00199
00200 /// <summary>
00201 /// Initializes a new instance of the <see cref="PDFRenderer"/> class.
00202 /// </summary>
00203         public PDFRenderer()
00204         {
00205             this.InitializeComponent();
00206
00207             this.PropertyChanged += ControlPropertyChanged;
00208             this.DetachedFromLogicalTree += ControlDetachedFromLogicalTree;
00209             this.PointerPressed += ControlPointerPressed;
00210             this.PointerReleased += ControlPointerReleased;
00211             this.PointerMoved += ControlPointerMoved;
00212             this.PointerWheelChanged += ControlPointerWheelChanged;
00213         }
00214
00215 /// <summary>
```

```
00216 /// Initializes inner components of the <see cref="PDFRenderer"/>.
00217 /// </summary>
00218         private void InitializeComponent()
00219         {
00220             PathFigure arrow1 = new PathFigure
00221             {
00222                 StartPoint = new Point(16 * Math.Cos(Math.PI / 4), 16 * Math.Sin(Math.PI / 4))
00223             };
00224             arrow1.Segments.Add(new ArcSegment() { Point = new Point(-16, 0), IsLargeArc = false,
      RotationAngle = 0, Size = new Avalonia.Size(16, 16), SweepDirection = SweepDirection.Clockwise });
00225             arrow1.Segments.Add(new LineSegment() { Point = new Point(-7.2727, 0) });
00226             arrow1.Segments.Add(new LineSegment() { Point = new Point(-21.8181, -17.4545) });
00227             arrow1.Segments.Add(new LineSegment() { Point = new Point(-36.3636, 0) });
00228             arrow1.Segments.Add(new LineSegment() { Point = new Point(-27.6363, 0) });
00229             arrow1.Segments.Add(new ArcSegment() { Point = new Point(27.6363 * Math.Cos(Math.PI / 4),
      27.6363 * Math.Sin(Math.PI / 4)), IsLargeArc = false, RotationAngle = 0, Size = new
      Avalonia.Size(27.6363, 27.6363), SweepDirection = SweepDirection.CounterClockwise });
00230             arrow1.IsClosed = true;
00231
00232             PathFigure arrow2 = new PathFigure
00233             {
00234                 StartPoint = new Point(16 * Math.Cos(5 * Math.PI / 4), 16 * Math.Sin(5 * Math.PI / 4))
00235             };
00236             arrow2.Segments.Add(new ArcSegment() { Point = new Point(16, 0), IsLargeArc = false,
      RotationAngle = 0, Size = new Avalonia.Size(16, 16), SweepDirection = SweepDirection.Clockwise });
00237             arrow2.Segments.Add(new LineSegment() { Point = new Point(7.2727, 0) });
00238             arrow2.Segments.Add(new LineSegment() { Point = new Point(21.8181, 17.4545) });
00239             arrow2.Segments.Add(new LineSegment() { Point = new Point(36.3636, 0) });
00240             arrow2.Segments.Add(new LineSegment() { Point = new Point(27.6363, 0) });
00241             arrow2.Segments.Add(new ArcSegment() { Point = new Point(27.6363 * Math.Cos(5 * Math.PI /
      4), 27.6363 * Math.Sin(5 * Math.PI / 4)), IsLargeArc = false, RotationAngle = 0, Size = new
      Avalonia.Size(27.6363, 27.6363), SweepDirection = SweepDirection.CounterClockwise });
00242             arrow2.IsClosed = true;
00243
00244             RefreshingGeometry = new PathGeometry();
00245             RefreshingGeometry.Figures.Add(arrow1);
00246             RefreshingGeometry.Figures.Add(arrow2);
00247         }
00248
00249 /// <summary>
00250 /// Set up the <see cref="PDFRenderer"/> to display a page of a <see cref="MuPDFDocument"/>.
00251 /// </summary>
00252 /// <param name="document">The <see cref="MuPDFDocument"/> to render.</param>
00253 /// <param name="threadCount">The number of threads to use in the rendering.  If this is 0, an
      appropriate number of threads based on the number of processors in the computer will be used.
      Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7.  If this is not the case, the
      biggest number smaller than <paramref name="threadCount"/> that satisfies this condition is
      used.</param>
00254 /// <param name="pageNumber">The index of the page that should be rendered.  The first page has index
      0.</param>
00255 /// <param name="resolutionMultiplier">This value can be used to increase or decrease the resolution
      at which the static renderisation of the page will be produced.  If <paramref
      name="resolutionMultiplier"/> is 1, the resolution will match the size (in screen units) of the <see
      cref="PDFRenderer"/>.</param>
00256 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included in the rendering.  Otherwise, only the page contents are included.</param>
00257 /// <param name="ocrLanguage">The language to use for optical character recognition (OCR). If this is
      null, no OCR is performed.</param>
00258         public void Initialize(MuPDFDocument document, int threadCount = 0, int pageNumber = 0, double
      resolutionMultiplier = 1, bool includeAnnotations = true, TesseractLanguage ocrLanguage = null)
00259         {
00260             if (IsViewerInitialized)
00261             {
00262                 ReleaseResources();
00263             }
00264
00265             OwnsContextAndDocument = false;
00266
00267             Document = document;
00268             Context = null;
00269
00270             ContinueInitialization(threadCount, pageNumber, resolutionMultiplier, includeAnnotations,
      ocrLanguage);
00271         }
00272
00273 /// <summary>
00274 /// Set up the <see cref="PDFRenderer"/> to display a page of a <see cref="MuPDFDocument"/>.  The OCR
      step is run asynchronously, in order not to block the UI thread.
00275 /// </summary>
00276 /// <param name="document">The <see cref="MuPDFDocument"/> to render.</param>
00277 /// <param name="threadCount">The number of threads to use in the rendering.  If this is 0, an
      appropriate number of threads based on the number of processors in the computer will be used.
      Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7.  If this is not the case, the
      biggest number smaller than <paramref name="threadCount"/> that satisfies this condition is
      used.</param>
00278 /// <param name="pageNumber">The index of the page that should be rendered.  The first page has index
      0.</param>
```

```
00279 /// <param name="resolutionMultiplier">This value can be used to increase or decrease the resolution
      at which the static renderisation of the page will be produced.  If <paramref
      name="resolutionMultiplier"/> is 1, the resolution will match the size (in screen units) of the <see
      cref="PDFRenderer"/>.</param>
00280 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included in the rendering.  Otherwise, only the page contents are included.</param>
00281 /// <param name="ocrLanguage">The language to use for optical character recognition (OCR). If this is
      null, no OCR is performed.</param>
00282 /// <param name="ocrCancellationToken">A <see cref="CancellationToken"/> used to cancel the OCR
      operation.</param>
00283 /// <param name="ocrProgress">An <see cref="IProgress{OCRProgressInfo}"/> used to report OCR
      progress.</param>
00284         public async Task InitializeAsync(MuPDFDocument document, int threadCount = 0, int pageNumber
      = 0, double resolutionMultiplier = 1, bool includeAnnotations = true, TesseractLanguage ocrLanguage =
      null, CancellationToken ocrCancellationToken = default, IProgress<OCRProgressInfo> ocrProgress = null)
00285         {
00286             if (IsViewerInitialized)
00287             {
00288                 ReleaseResources();
00289             }
00290
00291             OwnsContextAndDocument = false;
00292
00293             Document = document;
00294             Context = null;
00295
00296             await ContinueInitializationAsync(threadCount, pageNumber, resolutionMultiplier,
      includeAnnotations, ocrLanguage, ocrCancellationToken, ocrProgress);
00297         }
00298
00299 /// <summary>
00300 /// Set up the <see cref="PDFRenderer"/> to display a page of a document that will be loaded from
      disk.
00301 /// </summary>
00302 /// <param name="fileName">The path to the document that should be opened.</param>
00303 /// <param name="threadCount">The number of threads to use in the rendering.  If this is 0, an
      appropriate number of threads based on the number of processors in the computer will be used.
      Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7.  If this is not the case, the
      biggest number smaller than <paramref name="threadCount"/> that satisfies this condition is
      used.</param>
00304 /// <param name="pageNumber">The index of the page that should be rendered.  The first page has index
      0.</param>
00305 /// <param name="resolutionMultiplier">This value can be used to increase or decrease the resolution
      at which the static renderisation of the page will be produced.  If <paramref
      name="resolutionMultiplier"/> is 1, the resolution will match the size (in screen units) of the <see
      cref="PDFRenderer"/>.</param>
00306 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included in the rendering.  Otherwise, only the page contents are included.</param>
00307 /// <param name="ocrLanguage">The language to use for optical character recognition (OCR). If this is
      null, no OCR is performed.</param>
00308         public void Initialize(string fileName, int threadCount = 0, int pageNumber = 0, double
      resolutionMultiplier = 1, bool includeAnnotations = true, TesseractLanguage ocrLanguage = null)
00309         {
00310             if (IsViewerInitialized)
00311             {
00312                 ReleaseResources();
00313             }
00314
00315             OwnsContextAndDocument = true;
00316
00317             Context = new MuPDFContext();
00318             Document = new MuPDFDocument(Context, fileName);
00319
00320             ContinueInitialization(threadCount, pageNumber, resolutionMultiplier, includeAnnotations,
      ocrLanguage);
00321         }
00322
00323 /// <summary>
00324 /// Set up the <see cref="PDFRenderer"/> to display a page of a document that will be loaded from
      disk.  The OCR step is run asynchronously, in order not to block the UI thread.
00325 /// </summary>
00326 /// <param name="fileName">The path to the document that should be opened.</param>
00327 /// <param name="threadCount">The number of threads to use in the rendering.  If this is 0, an
      appropriate number of threads based on the number of processors in the computer will be used.
      Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7.  If this is not the case, the
      biggest number smaller than <paramref name="threadCount"/> that satisfies this condition is
      used.</param>
00328 /// <param name="pageNumber">The index of the page that should be rendered.  The first page has index
      0.</param>
00329 /// <param name="resolutionMultiplier">This value can be used to increase or decrease the resolution
      at which the static renderisation of the page will be produced.  If <paramref
      name="resolutionMultiplier"/> is 1, the resolution will match the size (in screen units) of the <see
      cref="PDFRenderer"/>.</param>
00330 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included in the rendering.  Otherwise, only the page contents are included.</param>
00331 /// <param name="ocrLanguage">The language to use for optical character recognition (OCR). If this is
      null, no OCR is performed.</param>
```

```
00332 /// <param name="ocrCancellationToken">A <see cref="CancellationToken"/> used to cancel the OCR
      operation.</param>
00333 /// <param name="ocrProgress">An <see cref="IProgress{OCRProgressInfo}"/> used to report OCR
      progress.</param>
00334         public async Task InitializeAsync(string fileName, int threadCount = 0, int pageNumber = 0,
      double resolutionMultiplier = 1, bool includeAnnotations = true, TesseractLanguage ocrLanguage = null,
      CancellationToken ocrCancellationToken = default, IProgress<OCRProgressInfo> ocrProgress = null)
00335         {
00336             if (IsViewerInitialized)
00337             {
00338                 ReleaseResources();
00339             }
00340 
00341             OwnsContextAndDocument = true;
00342 
00343             Context = new MuPDFContext();
00344             Document = new MuPDFDocument(Context, fileName);
00345 
00346             await ContinueInitializationAsync(threadCount, pageNumber, resolutionMultiplier,
      includeAnnotations, ocrLanguage, ocrCancellationToken, ocrProgress);
00347         }
00348 
00349 /// <summary>
00350 /// Set up the <see cref="PDFRenderer"/> to display a page of a document that will be loaded from a
      <see cref="MemoryStream"/>.
00351 /// </summary>
00352 /// <param name="ms">The <see cref="MemoryStream"/> containing the document that should be opened.
      This can be safely disposed after this method returns.</param>
00353 /// <param name="fileType">The format of the document.</param>
00354 /// <param name="threadCount">The number of threads to use in the rendering.  If this is 0, an
      appropriate number of threads based on the number of processors in the computer will be used.
      Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7.  If this is not the case, the
      biggest number smaller than <paramref name="threadCount"/> that satisfies this condition is
      used.</param>
00355 /// <param name="pageNumber">The index of the page that should be rendered.  The first page has index
      0.</param>
00356 /// <param name="resolutionMultiplier">This value can be used to increase or decrease the resolution
      at which the static renderisation of the page will be produced.  If <paramref
      name="resolutionMultiplier"/> is 1, the resolution will match the size (in screen units) of the <see
      cref="PDFRenderer"/>.</param>
00357 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included in the rendering.  Otherwise, only the page contents are included.</param>
00358 /// <param name="ocrLanguage">The language to use for optical character recognition (OCR). If this is
      null, no OCR is performed.</param>
00359         public void Initialize(MemoryStream ms, InputFileTypes fileType, int threadCount = 0, int
      pageNumber = 0, double resolutionMultiplier = 1, bool includeAnnotations = true, TesseractLanguage
      ocrLanguage = null)
00360         {
00361             //Get the byte array that underlies the MemoryStream.
00362             int origin = (int)ms.Seek(0, SeekOrigin.Begin);
00363             long dataLength = ms.Length;
00364             byte[] dataBytes = ms.GetBuffer();
00365 
00366             Initialize(dataBytes, fileType, origin, (int)dataLength, threadCount, pageNumber,
      resolutionMultiplier, includeAnnotations, ocrLanguage);
00367         }
00368 
00369 /// <summary>
00370 /// Set up the <see cref="PDFRenderer"/> to display a page of a document that will be loaded from a
      <see cref="MemoryStream"/>.  The OCR step is run asynchronously, in order not to block the UI thread.
00371 /// </summary>
00372 /// <param name="ms">The <see cref="MemoryStream"/> containing the document that should be opened.
      This can be safely disposed after this method returns.</param>
00373 /// <param name="fileType">The format of the document.</param>
00374 /// <param name="threadCount">The number of threads to use in the rendering.  If this is 0, an
      appropriate number of threads based on the number of processors in the computer will be used.
      Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7.  If this is not the case, the
      biggest number smaller than <paramref name="threadCount"/> that satisfies this condition is
      used.</param>
00375 /// <param name="pageNumber">The index of the page that should be rendered.  The first page has index
      0.</param>
00376 /// <param name="resolutionMultiplier">This value can be used to increase or decrease the resolution
      at which the static renderisation of the page will be produced.  If <paramref
      name="resolutionMultiplier"/> is 1, the resolution will match the size (in screen units) of the <see
      cref="PDFRenderer"/>.</param>
00377 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included in the rendering.  Otherwise, only the page contents are included.</param>
00378 /// <param name="ocrLanguage">The language to use for optical character recognition (OCR). If this is
      null, no OCR is performed.</param>
00379 /// <param name="ocrCancellationToken">A <see cref="CancellationToken"/> used to cancel the OCR
      operation.</param>
00380 /// <param name="ocrProgress">An <see cref="IProgress{OCRProgressInfo}"/> used to report OCR
      progress.</param>
00381         public async Task InitializeAsync(MemoryStream ms, InputFileTypes fileType, int threadCount =
      0, int pageNumber = 0, double resolutionMultiplier = 1, bool includeAnnotations = true,
      TesseractLanguage ocrLanguage = null, CancellationToken ocrCancellationToken = default,
      IProgress<OCRProgressInfo> ocrProgress = null)
```

```
00382         {
00383             //Get the byte array that underlies the MemoryStream.
00384             int origin = (int)ms.Seek(0, SeekOrigin.Begin);
00385             long dataLength = ms.Length;
00386             byte[] dataBytes = ms.GetBuffer();
00387 
00388             await InitializeAsync(dataBytes, fileType, origin, (int)dataLength, threadCount,
      pageNumber, resolutionMultiplier, includeAnnotations, ocrLanguage, ocrCancellationToken, ocrProgress);
00389         }
00390 
00391 /// <summary>
00392 /// Set up the <see cref="PDFRenderer"/> to display a page of a document that will be loaded from an
      array of <see cref="byte"/>s.
00393 /// </summary>
00394 /// <param name="dataBytes">The bytes of the document that should be opened.  The array will be copied
      and can be safely discarded/altered after this method returns.</param>
00395 /// <param name="fileType">The format of the document.</param>
00396 /// <param name="offset">The offset in the byte array at which the document starts.</param>
00397 /// <param name="length">The length of the document in bytes.  If this is &lt; 0, the whole array is
      used.</param>
00398 /// <param name="threadCount">The number of threads to use in the rendering.  If this is 0, an
      appropriate number of threads based on the number of processors in the computer will be used.
      Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7.  If this is not the case, the
      biggest number smaller than <paramref name="threadCount"/> that satisfies this condition is
      used.</param>
00399 /// <param name="pageNumber">The index of the page that should be rendered.  The first page has index
      0.</param>
00400 /// <param name="resolutionMultiplier">This value can be used to increase or decrease the resolution
      at which the static renderisation of the page will be produced.  If <paramref
      name="resolutionMultiplier"/> is 1, the resolution will match the size (in screen units) of the <see
      cref="PDFRenderer"/>.</param>
00401 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included in the rendering.  Otherwise, only the page contents are included.</param>
00402 /// <param name="ocrLanguage">The language to use for optical character recognition (OCR). If this is
      null, no OCR is performed.</param>
00403         public void Initialize(byte[] dataBytes, InputFileTypes fileType, int offset = 0, int length =
      -1, int threadCount = 0, int pageNumber = 0, double resolutionMultiplier = 1, bool includeAnnotations
      = true, TesseractLanguage ocrLanguage = null)
00404         {
00405             if (IsViewerInitialized)
00406             {
00407                 ReleaseResources();
00408             }
00409 
00410             if (length < 0)
00411             {
00412                 length = dataBytes.Length - offset;
00413             }
00414 
00415             //Copy the bytes to unmanaged memory, so that we don't depend on the original array.
00416             IntPtr pointer = Marshal.AllocHGlobal(length);
00417             Marshal.Copy(dataBytes, offset, pointer, length);
00418 
00419             //Wrap the pointer into a disposable container.
00420             IDisposable disposer = new DisposableIntPtr(pointer);
00421 
00422             OwnsContextAndDocument = true;
00423 
00424             Context = new MuPDFContext();
00425 
00426             //Create a new document, passing the wrapped pointer so that it can be released when the
      Document is disposed.
00427             Document = new MuPDFDocument(Context, pointer, length, fileType, ref disposer);
00428 
00429             ContinueInitialization(threadCount, pageNumber, resolutionMultiplier, includeAnnotations,
      ocrLanguage);
00430         }
00431 
00432 /// <summary>
00433 /// Set up the <see cref="PDFRenderer"/> to display a page of a document that will be loaded from an
      array of <see cref="byte"/>s.  The OCR step is run asynchronously, in order not to block the UI
      thread.
00434 /// </summary>
00435 /// <param name="dataBytes">The bytes of the document that should be opened.  The array will be copied
      and can be safely discarded/altered after this method returns.</param>
00436 /// <param name="fileType">The format of the document.</param>
00437 /// <param name="offset">The offset in the byte array at which the document starts.</param>
00438 /// <param name="length">The length of the document in bytes.  If this is &lt; 0, the whole array is
      used.</param>
00439 /// <param name="threadCount">The number of threads to use in the rendering.  If this is 0, an
      appropriate number of threads based on the number of processors in the computer will be used.
      Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7.  If this is not the case, the
      biggest number smaller than <paramref name="threadCount"/> that satisfies this condition is
      used.</param>
00440 /// <param name="pageNumber">The index of the page that should be rendered.  The first page has index
      0.</param>
00441 /// <param name="resolutionMultiplier">This value can be used to increase or decrease the resolution
```

```
      at which the static renderisation of the page will be produced.  If <paramref
      name="resolutionMultiplier"/> is 1, the resolution will match the size (in screen units) of the <see
      cref="PDFRenderer"/>.</param>
00442 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included in the rendering.  Otherwise, only the page contents are included.</param>
00443 /// <param name="ocrLanguage">The language to use for optical character recognition (OCR). If this is
      null, no OCR is performed.</param>
00444 /// <param name="ocrCancellationToken">A <see cref="CancellationToken"/> used to cancel the OCR
      operation.</param>
00445 /// <param name="ocrProgress">An <see cref="IProgress{OCRProgressInfo}"/> used to report OCR
      progress.</param>
00446         public async Task InitializeAsync(byte[] dataBytes, InputFileTypes fileType, int offset = 0,
      int length = -1, int threadCount = 0, int pageNumber = 0, double resolutionMultiplier = 1, bool
      includeAnnotations = true, TesseractLanguage ocrLanguage = null, CancellationToken
      ocrCancellationToken = default, IProgress<OCRProgressInfo> ocrProgress = null)
00447         {
00448             if (IsViewerInitialized)
00449             {
00450                 ReleaseResources();
00451             }
00452
00453             if (length < 0)
00454             {
00455                 length = dataBytes.Length - offset;
00456             }
00457
00458             //Copy the bytes to unmanaged memory, so that we don't depend on the original array.
00459             IntPtr pointer = Marshal.AllocHGlobal(length);
00460             Marshal.Copy(dataBytes, offset, pointer, length);
00461
00462             //Wrap the pointer into a disposable container.
00463             IDisposable disposer = new DisposableIntPtr(pointer);
00464
00465             OwnsContextAndDocument = true;
00466
00467             Context = new MuPDFContext();
00468
00469             //Create a new document, passing the wrapped pointer so that it can be released when the
      Document is disposed.
00470             Document = new MuPDFDocument(Context, pointer, length, fileType, ref disposer);
00471
00472             await ContinueInitializationAsync(threadCount, pageNumber, resolutionMultiplier,
      includeAnnotations, ocrLanguage, ocrCancellationToken, ocrProgress);
00473         }
00474
00475 /// <summary>
00476 /// Common steps in the initialization process that will be performed regardless of how the <see
      cref="Document"/> was obtained.
00477 /// </summary>
00478 /// <param name="threadCount">The number of threads to use in the rendering.  If this is 0, an
      appropriate number of threads based on the number of processors in the computer will be used.
      Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7.  If this is not the case, the
      biggest number smaller than <paramref name="threadCount"/> that satisfies this condition is
      used.</param>
00479 /// <param name="pageNumber">The index of the page that should be rendered.  The first page has index
      0.</param>
00480 /// <param name="resolutionMultiplier">This value can be used to increase or decrease the resolution
      at which the static renderisation of the page will be produced.  If <paramref
      name="resolutionMultiplier"/> is 1, the resolution will match the size (in screen units) of the <see
      cref="PDFRenderer"/>.</param>
00481 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included in the rendering.  Otherwise, only the page contents are included.</param>
00482 /// /// <param name="ocrLanguage">The language to use for optical character recognition (OCR). If this
      is null, no OCR is performed.</param>
00483         private void ContinueInitialization(int threadCount, int pageNumber, double
      resolutionMultiplier, bool includeAnnotations, TesseractLanguage ocrLanguage = null)
00484         {
00485             //Initialise threads and locking mechanics.
00486             if (RenderMutex == null)
00487             {
00488                 RenderMutex = new Mutex(false);
00489
00490                 this.RenderDynamicCanvasOuterThread = new Thread(() =>
00491                 {
00492                     RenderDynamicCanvasOuterAction();
00493                 });
00494
00495                 this.RenderDynamicCanvasInnerThread = new Thread(() =>
00496                 {
00497                     RenderDynamicCanvasInnerAction();
00498                 });
00499
00500                 RenderDynamicCanvasOuterThread.Start();
00501                 RenderDynamicCanvasInnerThread.Start();
00502             }
00503
00504             //Choose an appropriate number of threads based on the number of processors in the
```

```
computer.  We have an upper limit of 8 threads because more threads apparently caused reduced
performance due to the synchronisation overhead.
00505             if (threadCount <= 0)
00506             {
00507                 threadCount = Math.Max(1, Math.Min(8, Environment.ProcessorCount - 2));
00508             }
00509 
00510             //Create the structured text representation.
00511             this.StructuredTextPage = Document.GetStructuredTextPage(pageNumber, ocrLanguage,
includeAnnotations);
00512 
00513             //Create the multithreaded renderer.
00514             Renderer = Document.GetMultiThreadedRenderer(pageNumber, threadCount, includeAnnotations);
00515 
00516             //Set up the properties of this control.
00517             RenderThreadCount = Renderer.ThreadCount;
00518             Rectangle bounds = Document.Pages[pageNumber].Bounds;
00519             PageSize = new Rect(new Point(bounds.X0, bounds.Y0), new Point(bounds.X1, bounds.Y1));
00520             PageNumber = pageNumber;
00521 
00522             //Render the static canvas (which is used when the DynamicBitmaps are not available).
00523             RenderFixedCanvas(resolutionMultiplier);
00524 
00525             //Initialize the dynamic canvas.
00526             InitializeDynamicCanvas();
00527 
00528             //Set initial display area to include the whole page.
00529             double widthRatio = PageSize.Width / (this.Bounds.Width * resolutionMultiplier);
00530             double heightRatio = PageSize.Height / (this.Bounds.Height * resolutionMultiplier);
00531 
00532             double containingWidth = Math.Max(widthRatio, heightRatio) * this.Bounds.Width *
resolutionMultiplier;
00533             double containingHeight = Math.Max(widthRatio, heightRatio) * this.Bounds.Height *
resolutionMultiplier;
00534 
00535             SetDisplayAreaNowInternal(new Rect(new Point(-(containingWidth - FixedArea.Width) * 0.5,
-(containingHeight - FixedArea.Height) * 0.5), new Avalonia.Size(containingWidth, containingHeight)));
00536             this._Zoom = this.Bounds.Width / DisplayArea.Width * 72 / 96 * (VisualRoot as
ILayoutRoot).LayoutScaling;
00537 
00538             //We are ready!
00539             IsViewerInitialized = true;
00540 
00541             //Queue a render of the DynamicBitmaps (on another thread).
00542             RenderDynamicCanvas();
00543         }
00544 
00545 
00546 /// <summary>
00547 /// Common steps in the initialization process that will be performed regardless of how the <see
cref="Document"/> was obtained.  The OCR step is run asynchronously, in order not to block the UI
thread.
00548 /// </summary>
00549 /// <param name="threadCount">The number of threads to use in the rendering.  If this is 0, an
appropriate number of threads based on the number of processors in the computer will be used.
Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7.  If this is not the case, the
biggest number smaller than <paramref name="threadCount"/> that satisfies this condition is
used.</param>
00550 /// <param name="pageNumber">The index of the page that should be rendered.  The first page has index
0.</param>
00551 /// <param name="resolutionMultiplier">This value can be used to increase or decrease the resolution
at which the static renderisation of the page will be produced.  If <paramref
name="resolutionMultiplier"/> is 1, the resolution will match the size (in screen units) of the <see
cref="PDFRenderer"/>.</param>
00552 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
signatures) are included in the rendering.  Otherwise, only the page contents are included.</param>
00553 /// <param name="ocrLanguage">The language to use for optical character recognition (OCR). If this is
null, no OCR is performed.</param>
00554 /// <param name="ocrCancellationToken">A <see cref="CancellationToken"/> used to cancel the OCR
operation.</param>
00555 /// <param name="ocrProgress">An <see cref="IProgress{OCRProgressInfo}"/> used to report OCR
progress.</param>
00556         private async Task ContinueInitializationAsync(int threadCount, int pageNumber, double
resolutionMultiplier, bool includeAnnotations, TesseractLanguage ocrLanguage, CancellationToken
ocrCancellationToken, IProgress<OCRProgressInfo> ocrProgress)
00557         {
00558             //Initialise threads and locking mechanics.
00559             if (RenderMutex == null)
00560             {
00561                 RenderMutex = new Mutex(false);
00562 
00563                 this.RenderDynamicCanvasOuterThread = new Thread(() =>
00564                 {
00565                     RenderDynamicCanvasOuterAction();
00566                 });
00567 
00568                 this.RenderDynamicCanvasInnerThread = new Thread(() =>
```

```
00569                 {
00570                     RenderDynamicCanvasInnerAction();
00571                 });
00572 
00573                 RenderDynamicCanvasOuterThread.Start();
00574                 RenderDynamicCanvasInnerThread.Start();
00575             }
00576 
00577             //Choose an appropriate number of threads based on the number of processors in the computer.  We have an upper limit of 8 threads because more threads apparently caused reduced performance due to the synchronisation overhead.
00578             if (threadCount <= 0)
00579             {
00580                 threadCount = Math.Max(1, Math.Min(8, Environment.ProcessorCount - 2));
00581             }
00582 
00583             //Create the structured text representation.
00584             this.StructuredTextPage = await Document.GetStructuredTextPageAsync(pageNumber, ocrLanguage, includeAnnotations, ocrCancellationToken, ocrProgress);
00585 
00586             //Create the multithreaded renderer.
00587             Renderer = Document.GetMultiThreadedRenderer(pageNumber, threadCount, includeAnnotations);
00588 
00589             //Set up the properties of this control.
00590             RenderThreadCount = Renderer.ThreadCount;
00591             Rectangle bounds = Document.Pages[pageNumber].Bounds;
00592             PageSize = new Rect(new Point(bounds.X0, bounds.Y0), new Point(bounds.X1, bounds.Y1));
00593             PageNumber = pageNumber;
00594 
00595             //Render the static canvas (which is used when the DynamicBitmaps are not available).
00596             RenderFixedCanvas(resolutionMultiplier);
00597 
00598             //Initialize the dynamic canvas.
00599             InitializeDynamicCanvas();
00600 
00601             //Set initial display area to include the whole page.
00602             double widthRatio = PageSize.Width / (this.Bounds.Width * resolutionMultiplier);
00603             double heightRatio = PageSize.Height / (this.Bounds.Height * resolutionMultiplier);
00604 
00605             double containingWidth = Math.Max(widthRatio, heightRatio) * this.Bounds.Width * resolutionMultiplier;
00606             double containingHeight = Math.Max(widthRatio, heightRatio) * this.Bounds.Height * resolutionMultiplier;
00607 
00608             SetDisplayAreaNowInternal(new Rect(new Point(-(containingWidth - FixedArea.Width) * 0.5, -(containingHeight - FixedArea.Height) * 0.5), new Avalonia.Size(containingWidth, containingHeight)));
00609             this._Zoom = this.Bounds.Width / DisplayArea.Width * 72 / 96 * (VisualRoot as ILayoutRoot).LayoutScaling;
00610 
00611             //We are ready!
00612             IsViewerInitialized = true;
00613 
00614             //Queue a render of the DynamicBitmaps (on another thread).
00615             RenderDynamicCanvas();
00616         }
00617 
00618 /// <summary>
00619 /// Release resources held by this PDFRenderer.  This is not an irreversible step:  using one of the Initialize overloads after calling this method will restore functionality.
00620 /// </summary>
00621         public void ReleaseResources()
00622         {
00623             IsViewerInitialized = false;
00624             this.Renderer?.Dispose();
00625             this.StructuredTextPage = null;
00626             this.Selection = null;
00627             this.HighlightedRegions = null;
00628 
00629             if (OwnsContextAndDocument)
00630             {
00631                 this.Document?.Dispose();
00632                 this.Context?.Dispose();
00633             }
00634         }
00635 
00636 /// <summary>
00637 /// Called when the PDFRenderer is removed from the logical tree (e.g.  it is removed from the window, or the window containing it is closed).  We assume that this renderer is not needed anymore.  This is irreversible!
00638 /// </summary>
00639         private void ControlDetachedFromLogicalTree(object sender, LogicalTreeAttachmentEventArgs e)
00640         {
00641             RendererDisposedHandle.Set();
00642             ReleaseResources();
00643         }
00644 
00645 
```

```
00646 /// <summary>
00647 /// Set the current display area to the specified <paramref name="value"/>, skipping all transitions.
      This also skips sanity checks of the <paramref name="value"/>, since the calling methods will already
      have performed them.
00648 /// </summary>
00649 /// <param name="value">The new display area.</param>
00650         private void SetDisplayAreaNowInternal(Rect value)
00651         {
00652             Transitions prevTransitions = this.Transitions;
00653             this.Transitions = null;
00654             SetValue(DisplayAreaProperty, value);
00655             this.Transitions = prevTransitions;
00656         }
00657 
00658 /// <summary>
00659 /// Set the current display area to the specified <paramref name="value"/>, skipping all transitions.
00660 /// </summary>
00661 /// <param name="value">The new display area.</param>
00662         public void SetDisplayAreaNow(Rect value)
00663         {
00664             Transitions prevTransitions = this.Transitions;
00665             this.Transitions = null;
00666             this.DisplayArea = value;
00667             this.Transitions = prevTransitions;
00668         }
00669 
00670 /// <summary>
00671 /// Zoom around a point.
00672 /// </summary>
00673 /// <param name="count">Number of steps to zoom.  Positive values indicate a zoom in, negative values
      a zoom out.</param>
00674 /// <param name="center">The point around which to center the zoom operation.  If this is null, the
      center of the control is used.</param>
00675         public void ZoomStep(double count, Point?  center = null)
00676         {
00677             if (center == null)
00678             {
00679                 center = new Point(this.Bounds.Width * 0.5, this.Bounds.Height * 0.5);
00680             }
00681 
00682             double currZoomX = FixedArea.Width / DisplayArea.Width;
00683             double currZoomY = FixedArea.Height / DisplayArea.Height;
00684 
00685             currZoomX *= Math.Pow(ZoomIncrement, count);
00686             currZoomY *= Math.Pow(ZoomIncrement, count);
00687 
00688             double currWidth = FixedArea.Width / currZoomX;
00689             double currHeight = FixedArea.Height / currZoomY;
00690 
00691             double deltaW = currWidth - DisplayArea.Width;
00692             double deltaH = currHeight - DisplayArea.Height;
00693 
00694             SetValue(DisplayAreaProperty, new Rect(new Point(DisplayArea.X - deltaW * center.Value.X /
      this.Bounds.Width, DisplayArea.Y - deltaH * center.Value.Y / this.Bounds.Height), new
      Point(DisplayArea.Right + deltaW * (1 - center.Value.X / this.Bounds.Width), DisplayArea.Bottom +
      deltaH * (1 - center.Value.Y / this.Bounds.Height))));
00695         }
00696 
00697 /// <summary>
00698 /// Alter the display area so that the whole page fits on screen.
00699 /// </summary>
00700         public void Contain()
00701         {
00702             //This will be sanitised by the property setter.
00703             this.DisplayArea = this.PageSize;
00704         }
00705 
00706 /// <summary>
00707 /// Alter the display area so that the page covers the whole surface of the <see cref="PDFRenderer"/>
      (even though parts of the page may be outside it).
00708 /// </summary>
00709         public void Cover()
00710         {
00711             double widthRatio = this.PageSize.Width / (this.Bounds.Width);
00712             double heightRatio = this.PageSize.Height / (this.Bounds.Height);
00713 
00714             double containingWidth = Math.Min(widthRatio, heightRatio) * this.Bounds.Width;
00715             double containingHeight = Math.Min(widthRatio, heightRatio) * this.Bounds.Height;
00716 
00717             double deltaW = (containingWidth - this.PageSize.Width) * 0.5;
00718             double deltaH = (containingHeight - this.PageSize.Height) * 0.5;
00719 
00720             Rect newDispArea = new Rect(new Point(this.PageSize.X - deltaW, this.PageSize.Y - deltaH),
      new Point(this.PageSize.Right + deltaW, this.PageSize.Bottom + deltaH));
00721 
00722             //Skip sanitation.
00723             SetValue(DisplayAreaProperty, newDispArea);
```

```
00724         }
00725
00726 /// <summary>
00727 /// Get the current rendering progress.
00728 /// </summary>
00729 /// <returns>A <see cref="RenderProgress"/> object with information about the rendering progress of
      each thread.</returns>
00730         public RenderProgress GetProgress()
00731         {
00732             return Renderer.GetProgress();
00733         }
00734
00735 /// <summary>
00736 /// Get the currently selected text.
00737 /// </summary>
00738 /// <returns>The currently selected text.</returns>
00739         public string GetSelectedText()
00740         {
00741             return this.StructuredTextPage.GetText(this.Selection);
00742         }
00743
00744 /// <summary>
00745 /// Selects all the text in the document.
00746 /// </summary>
00747         public void SelectAll()
00748         {
00749             if (this.StructuredTextPage.Count > 0)
00750             {
00751                 int maxBlock = this.StructuredTextPage.Count - 1;
00752                 int maxLine = this.StructuredTextPage[maxBlock].Count - 1;
00753                 int maxCharacter = this.StructuredTextPage[maxBlock][maxLine].Count - 1;
00754
00755                 this.Selection = new MuPDFStructuredTextAddressSpan(new MuPDFStructuredTextAddress(0,
      0, 0), new MuPDFStructuredTextAddress(maxBlock, maxLine, maxCharacter));
00756             }
00757             else
00758             {
00759                 this.Selection = null;
00760             }
00761         }
00762
00763 /// <summary>
00764 /// Highlights all matches of the specified <see cref="Regex"/> in the text and returns the number of
      matches found.  Matches cannot span multiple lines.
00765 /// </summary>
00766 /// <param name="needle">The <see cref="Regex"/> to search for.</param>
00767 /// <returns>The number of matches that have been found.</returns>
00768         public int Search(Regex needle)
00769         {
00770             List<MuPDFStructuredTextAddressSpan> spans =
      this.StructuredTextPage.Search(needle).ToList();
00771             this.HighlightedRegions = spans;
00772             return spans.Count;
00773         }
00774
00775 /// <summary>
00776 /// Render the <see cref="FixedCanvasBitmap"/>.
00777 /// </summary>
00778 /// <param name="resolutionMultiplier">This value can be used to increase or decrease the resolution
      at which the static renderisation of the page will be produced.  If <paramref
      name="resolutionMultiplier"/> is 1, the resolution will match the size (in screen units) of the <see
      cref="PDFRenderer"/>.</param>
00779         private void RenderFixedCanvas(double resolutionMultiplier)
00780         {
00781             //Take into account DPI scaling.
00782             resolutionMultiplier *= (VisualRoot as ILayoutRoot)?.LayoutScaling ??  1;
00783
00784             double widthRatio = PageSize.Width / (this.Bounds.Width * resolutionMultiplier);
00785             double heightRatio = PageSize.Height / (this.Bounds.Height * resolutionMultiplier);
00786
00787             double zoom = 1 / Math.Min(widthRatio, heightRatio);
00788
00789             //Render the whole page
00790             Rectangle origin = new Rectangle(0, 0, PageSize.Width, PageSize.Height);
00791
00792             FixedArea = origin;
00793
00794             RoundedRectangle roundedOrigin = origin.Round(zoom);
00795
00796             RoundedSize targetSize = new RoundedSize(roundedOrigin.Width, roundedOrigin.Height);
00797             if (FixedCanvasBitmap == null)
00798             {
00799                 FixedCanvasBitmap = new WriteableBitmap(new PixelSize(targetSize.Width,
      targetSize.Height), new Vector(72, 72), Avalonia.Platform.PixelFormat.Rgba8888, AlphaFormat.Unpremul);
00800             }
00801             else
00802             {
```

```
00803                 if (FixedCanvasBitmap.PixelSize.Width != targetSize.Width ||
      FixedCanvasBitmap.PixelSize.Height != targetSize.Height)
00804                 {
00805                     FixedCanvasBitmap = new WriteableBitmap(new PixelSize(targetSize.Width,
      targetSize.Height), new Vector(72, 72), Avalonia.Platform.PixelFormat.Rgba8888, AlphaFormat.Unpremul);
00806                 }
00807             }
00808 
00809             //Render the page to the FixedCanvasBitmap (without marshaling).
00810             using (ILockedFramebuffer fb = FixedCanvasBitmap.Lock())
00811             {
00812                 Document.Render(PageNumber, origin, zoom, PixelFormats.RGBA, fb.Address);
00813             }
00814         }
00815 
00816 /// <summary>
00817 /// Set up the <see cref="DynamicBitmaps"/> array with an appropriate number of <see
      cref="WriteableBitmap"/> of the appropriate size.
00818 /// </summary>
00819         private void InitializeDynamicCanvas()
00820         {
00821             //Take into account DPI scaling.
00822             double scale = (VisualRoot as ILayoutRoot)?.LayoutScaling ??  1;
00823 
00824             //Acquire the render mutex (we don't want anyone to touch the DynamicBitmaps while we are
      resizing them!)
00825             RenderMutex.WaitOne();
00826             RoundedSize targetSize = new RoundedSize((int)Math.Ceiling(this.Bounds.Width * scale),
      (int)Math.Ceiling(this.Bounds.Height * scale));
00827 
00828             //Split the target size into an appropriate number of tiles.
00829             RoundedRectangle[] splitSizes = targetSize.Split(RenderThreadCount);
00830 
00831             DynamicImagesBounds = splitSizes;
00832 
00833             if (DynamicBitmaps == null || DynamicBitmaps.Length != RenderThreadCount)
00834             {
00835                 DynamicBitmaps = new WriteableBitmap[RenderThreadCount];
00836                 for (int i = 0; i < splitSizes.Length; i++)
00837                 {
00838                     DynamicBitmaps[i] = new WriteableBitmap(new PixelSize(splitSizes[i].Width,
      splitSizes[i].Height), new Vector(72, 72), Avalonia.Platform.PixelFormat.Rgba8888,
      AlphaFormat.Unpremul);
00839                 }
00840             }
00841             else
00842             {
00843                 for (int i = 0; i < splitSizes.Length; i++)
00844                 {
00845                     if (DynamicBitmaps[i].PixelSize.Width != splitSizes[i].Width ||
      DynamicBitmaps[i].PixelSize.Height != splitSizes[i].Height)
00846                     {
00847                         DynamicBitmaps[i] = new WriteableBitmap(new PixelSize(splitSizes[i].Width,
      splitSizes[i].Height), new Vector(72, 72), Avalonia.Platform.PixelFormat.Rgba8888,
      AlphaFormat.Unpremul);
00848                     }
00849                 }
00850             }
00851 
00852             //Release the render mutex.
00853             RenderMutex.ReleaseMutex();
00854         }
00855 
00856 /// <summary>
00857 /// The outer loop that is executed by the <see cref="RenderDynamicCanvasOuterThread"/>, which is in
      charge of responding to the rendering requests and either starting a new rendering of the <see
      cref="DynamicBitmaps"/>, or queueing it.
00858 /// </summary>
00859         private void RenderDynamicCanvasOuterAction()
00860         {
00861             EventWaitHandle[] handles = new EventWaitHandle[] { RenderDynamicCanvasOuterHandle,
      RendererDisposedHandle };
00862 
00863             while (true)
00864             {
00865                 int result = EventWaitHandle.WaitAny(handles);
00866 
00867                 if (result == 0)
00868                 {
00869                     //So that we don't lose consecutive requests.
00870                     RenderDynamicCanvasOuterHandle.Reset();
00871 
00872                     //Check if the rendering is already in progress.
00873                     if (RenderMutex.WaitOne(0))
00874                     {
00875                         //Start a new rendering
00876                         AreDynamicBitmapsReady = false;
```

```
00877
00878                       //This handle will be set by the inner thread once it starts rendering.
00879                       RenderDynamicCanvasInnerStartedHandle.Reset();
00880
00881                       //Tell the inner thread to start rendering.
00882                       RenderDynamicCanvasInnerHandle.Set();
00883
00884                       //Release the mutex so that the inner thread can start rendering.
00885                       RenderMutex.ReleaseMutex();
00886
00887                       //Wait until the inner thread has acuired the mutex and started rendering.
00888                       RenderDynamicCanvasInnerStartedHandle.WaitOne();
00889                   }
00890                   else
00891                   {
00892                       if (!RenderQueued)
00893                       {
00894                           //Queue another rendering pass.
00895                           RenderQueued = true;
00896
00897                           //Abort the current rendering pass.
00898                           Renderer.Abort();
00899                       }
00900                   }
00901               }
00902               else
00903               {
00904                   //The renderer is being disposed, we need to quit!
00905                   break;
00906               }
00907           }
00908       }
00909
00910 /// <summary>
00911 /// The inner loop that is executed by the <see cref="RenderDynamicCanvasInnerThread"/>, which renders
      the <see cref="DynamicBitmaps"/>.
00912 /// </summary>
00913       private void RenderDynamicCanvasInnerAction()
00914       {
00915           EventWaitHandle[] handles = new EventWaitHandle[] { RenderDynamicCanvasInnerHandle,
      RendererDisposedHandle };
00916
00917           bool ownsMutex = false;
00918
00919           while (true)
00920           {
00921               int result = EventWaitHandle.WaitAny(handles);
00922
00923               if (result == 0)
00924               {
00925                   //So that we don't lose consecutive requests.
00926                   RenderDynamicCanvasInnerHandle.Reset();
00927
00928                   //Acquire the mutex only if have not acquired it yet.
00929                   if (!ownsMutex)
00930                   {
00931                       RenderMutex.WaitOne();
00932                       ownsMutex = true;
00933                   }
00934
00935                   //Signal to the outer thread that we have acquired the mutex.  Even if the outer
      thread is not waiting for this signal, it will reset it before waiting for it.
00936                   RenderDynamicCanvasInnerStartedHandle.Set();
00937
00938                   //Set up the pointers to the contents of the DynamicBitmaps
00939                   IntPtr[] destinations = new IntPtr[RenderThreadCount];
00940                   ILockedFramebuffer[] fbs = new ILockedFramebuffer[RenderThreadCount];
00941
00942                   for (int i = 0; i < RenderThreadCount; i++)
00943                   {
00944                       fbs[i] = DynamicBitmaps[i].Lock();
00945                       destinations[i] = fbs[i].Address;
00946                   }
00947
00948                   //Prevent race conditions.
00949                   Rectangle target;
00950                   int width;
00951                   int height;
00952                   lock (RenderDisplayAreaLock)
00953                   {
00954                       target = new Rectangle(RenderDisplayArea.X, RenderDisplayArea.Y,
      RenderDisplayArea.Right, RenderDisplayArea.Bottom);
00955                       width = RenderSize[0];
00956                       height = RenderSize[1];
00957                   }
00958
00959                   //Start the multithreaded rendering and wait until it finishes.
```

```
00960                     Renderer.Render(new RoundedSize(width, height), target, destinations,
      PixelFormats.RGBA);
00961
00962                     //Free the pointers.
00963                     for (int i = 0; i < RenderThreadCount; i++)
00964                     {
00965                         fbs[i].Dispose();
00966                     }
00967
00968                     //Check whether another rendering request has been queued.
00969                     if (!RenderQueued)
00970                     {
00971                         //No other rendering requests have been queued.
00972                         AreDynamicBitmapsReady = true;
00973
00974                         //This should always be true.  Release the rendering mutex.
00975                         if (ownsMutex)
00976                         {
00977                             RenderMutex.ReleaseMutex();
00978                             ownsMutex = false;
00979                         }
00980
00981                         //Signal to the UI that a repaint is required.
00982                         Dispatcher.UIThread.InvokeAsync(() =>
00983                         {
00984                             this.InvalidateVisual();
00985                         });
00986                     }
00987                     else
00988                     {
00989                         //Another rendering request has been queued, we can assume that whatever we
      have rendered until now is useless (maybe because the rendering has been aborted).
00990                         RenderQueued = false;
00991
00992                         //Self-signal.
00993                         RenderDynamicCanvasInnerHandle.Set();
00994                     }
00995                 }
00996                 else
00997                 {
00998                     //The renderer is being disposed, we need to quit!
00999                     break;
01000                 }
01001             }
01002         }
01003
01004 /// <summary>
01005 /// Signal to the <see cref="RenderDynamicCanvasOuterThread"/> that a rendering has been requested.
01006 /// </summary>
01007         private void RenderDynamicCanvas()
01008         {
01009             //Take into account DPI scaling.
01010             double scale = (VisualRoot as ILayoutRoot).LayoutScaling;
01011
01012             //Set up rendering size
01013             lock (RenderDisplayAreaLock)
01014             {
01015                 RenderSize[0] = (int)Math.Ceiling(this.Bounds.Width * scale);
01016                 RenderSize[1] = (int)Math.Ceiling(this.Bounds.Height * scale);
01017                 RenderDisplayArea = DisplayArea;
01018             }
01019
01020             //Send the signal.
01021             RenderDynamicCanvasOuterHandle.Set();
01022         }
01023
01024 /// <summary>
01025 /// Resizes the <see cref="DynamicBitmaps"/> when the size of the control changes and queues a repaint
      when the <see cref="DisplayArea"/> changes.
01026 /// </summary>
01027 /// <param name="sender"></param>
01028 /// <param name="e"></param>
01029         private void ControlPropertyChanged(object sender, AvaloniaPropertyChangedEventArgs e)
01030         {
01031             if (e.Property == UserControl.BoundsProperty)
01032             {
01033                 if (IsViewerInitialized)
01034                 {
01035                     //Resize the display area
01036                     Rect oldBounds = (Rect)e.OldValue;
01037                     Rect newBounds = (Rect)e.NewValue;
01038
01039                     double deltaW = (newBounds.Width - oldBounds.Width) / oldBounds.Width *
      DisplayArea.Width;
01040                     double deltaH = (newBounds.Height - oldBounds.Height) / oldBounds.Height *
      DisplayArea.Height;
01041
```

```
01042                     Rect target = new Rect(new Point(DisplayArea.X - deltaW * 0.5, DisplayArea.Y -
      deltaH * 0.5), new Point(DisplayArea.Right + deltaW * 0.5, DisplayArea.Bottom + deltaH * 0.5));
01043
01044                     //Resize the DynamicBitmaps
01045                     InitializeDynamicCanvas();
01046
01047                     //Set the new DisplayArea, skipping any animation.
01048                     SetDisplayAreaNowInternal(target);
01049                 }
01050             }
01051             else if (e.Property == PDFRenderer.DisplayAreaProperty)
01052             {
01053                 if (IsViewerInitialized)
01054                 {
01055                     //Update the value of the Zoom property.
01056                     ComputeZoom();
01057
01058                     //Signal that a repaint is needed
01059                     this.InvalidateVisual();
01060
01061                     //Queue a new rendering of the DynamicBitmaps
01062                     RenderDynamicCanvas();
01063                 }
01064             }
01065             else if (e.Property == PDFRenderer.SelectionProperty && this.StructuredTextPage != null)
01066             {
01067                 //Update the selection quads to reflect the new selection
01068                 this.SelectionQuads = this.StructuredTextPage.GetHighlightQuads(this.Selection,
      false).ToList();
01069                 this.InvalidateVisual();
01070             }
01071             else if (e.Property == PDFRenderer.HighlightedRegionsProperty && this.StructuredTextPage
      != null)
01072             {
01073                 //Update the highlight quads to reflect the new highlighted regions
01074                 this.HighlightQuads = new List<Quad>();
01075
01076                 if (this.HighlightedRegions != null)
01077                 {
01078                     foreach (MuPDFStructuredTextAddressSpan span in this.HighlightedRegions)
01079                     {
01080                         this.HighlightQuads.AddRange(this.StructuredTextPage.GetHighlightQuads(span,
      false));
01081                     }
01082                 }
01083
01084                 this.InvalidateVisual();
01085             }
01086         }
01087
01088 /// <summary>
01089 /// Default handler for the PointerPressed event (start panning).
01090 /// </summary>
01091 /// <param name="sender"></param>
01092 /// <param name="e"></param>
01093         private void ControlPointerPressed(object sender, PointerPressedEventArgs e)
01094         {
01095             if (PointerEventHandlersType == PointerEventHandlers.Pan)
01096             {
01097                 if (e.GetCurrentPoint(this).Properties.PointerUpdateKind ==
      PointerUpdateKind.LeftButtonPressed)
01098                 {
01099                     IsMouseDown = true;
01100                     MouseDownPoint = e.GetPosition(this);
01101                     MouseDownDisplayArea = DisplayArea;
01102                     this.Cursor = new Cursor(StandardCursorType.SizeAll);
01103                 }
01104             }
01105             else if (PointerEventHandlersType == PointerEventHandlers.Highlight)
01106             {
01107                 if (e.GetCurrentPoint(this).Properties.PointerUpdateKind ==
      PointerUpdateKind.LeftButtonPressed)
01108                 {
01109                     Point point = e.GetPosition(this);
01110
01111                     IsMouseDown = true;
01112                     MouseDownPoint = point;
01113                     MouseDownDisplayArea = DisplayArea;
01114
01115                     PointF pagePoint = new PointF((float)(point.X / this.Bounds.Width *
      DisplayArea.Width + DisplayArea.Left), (float)(point.Y / this.Bounds.Height * DisplayArea.Height +
      DisplayArea.Top));
01116
01117                     MuPDFStructuredTextAddress?  address =
      StructuredTextPage?.GetHitAddress(pagePoint, false);
01118
01119                     if (address != null)
```

```
01120                     {
01121                         this.Selection = new MuPDFStructuredTextAddressSpan(address.Value, null);
01122                     }
01123                     else
01124                     {
01125                         this.Selection = null;
01126                     }
01127                 }
01128             }
01129             else if (PointerEventHandlersType == PointerEventHandlers.PanHighlight)
01130             {
01131                 Point point = e.GetPosition(this);
01132                 PointF pagePoint = new PointF((float)(point.X / this.Bounds.Width * DisplayArea.Width
      + DisplayArea.Left), (float)(point.Y / this.Bounds.Height * DisplayArea.Height + DisplayArea.Top));
01133                 MuPDFStructuredTextAddress?  address = StructuredTextPage?.GetHitAddress(pagePoint,
      false);
01134 
01135                 if (address == null)
01136                 {
01137                     if (e.GetCurrentPoint(this).Properties.PointerUpdateKind ==
      PointerUpdateKind.LeftButtonPressed)
01138                     {
01139                         IsMouseDown = true;
01140                         MouseDownPoint = e.GetPosition(this);
01141                         MouseDownDisplayArea = DisplayArea;
01142                         this.Cursor = new Cursor(StandardCursorType.SizeAll);
01143                         CurrentMouseOperation = CurrentMouseOperations.Pan;
01144                     }
01145                 }
01146                 else
01147                 {
01148                     if (e.GetCurrentPoint(this).Properties.PointerUpdateKind ==
      PointerUpdateKind.LeftButtonPressed)
01149                     {
01150                         IsMouseDown = true;
01151                         MouseDownPoint = point;
01152                         MouseDownDisplayArea = DisplayArea;
01153 
01154                         this.Selection = new MuPDFStructuredTextAddressSpan(address.Value, null);
01155                         CurrentMouseOperation = CurrentMouseOperations.Highlight;
01156                     }
01157                 }
01158             }
01159         }
01160 
01161 /// <summary>
01162 /// Default handler for the PointerReleased event (finish panning).
01163 /// </summary>
01164 /// <param name="sender"></param>
01165 /// <param name="e"></param>
01166         private void ControlPointerReleased(object sender, PointerReleasedEventArgs e)
01167         {
01168             if (PointerEventHandlersType == PointerEventHandlers.Pan)
01169             {
01170                 if (e.InitialPressMouseButton == MouseButton.Left)
01171                 {
01172                     IsMouseDown = false;
01173                     this.Cursor = new Cursor(StandardCursorType.Arrow);
01174                 }
01175             }
01176             else if (PointerEventHandlersType == PointerEventHandlers.Highlight)
01177             {
01178                 if (e.InitialPressMouseButton == MouseButton.Left)
01179                 {
01180                     IsMouseDown = false;
01181                     if (e.GetPosition(this).Equals(MouseDownPoint))
01182                     {
01183                         this.Selection = null;
01184                     }
01185                 }
01186             }
01187             else if (PointerEventHandlersType == PointerEventHandlers.PanHighlight)
01188             {
01189                 if (e.InitialPressMouseButton == MouseButton.Left)
01190                 {
01191                     IsMouseDown = false;
01192                     if (CurrentMouseOperation == CurrentMouseOperations.Pan)
01193                     {
01194                         this.Cursor = new Cursor(StandardCursorType.Arrow);
01195                     }
01196                     if (e.GetPosition(this).Equals(MouseDownPoint))
01197                     {
01198                         this.Selection = null;
01199                     }
01200                 }
01201             }
01202         }
```

```
01203
01204 /// <summary>
01205 /// Default handler for the PointerMoved event (pan).
01206 /// </summary>
01207 /// <param name="sender"></param>
01208 /// <param name="e"></param>
01209         private void ControlPointerMoved(object sender, PointerEventArgs e)
01210         {
01211             if (IsMouseDown)
01212             {
01213                 if (PointerEventHandlersType == PointerEventHandlers.Pan || (PointerEventHandlersType
      == PointerEventHandlers.PanHighlight && CurrentMouseOperation == CurrentMouseOperations.Pan))
01214                 {
01215
01216                     Point point = e.GetPosition(this);
01217
01218                     double deltaX = (-point.X + MouseDownPoint.X) / this.Bounds.Width *
      DisplayArea.Width;
01219                     double deltaY = (-point.Y + MouseDownPoint.Y) / this.Bounds.Height *
      DisplayArea.Height;
01220
01221                     Rect target = new Rect(new Point(this.MouseDownDisplayArea.X + deltaX,
      this.MouseDownDisplayArea.Y + deltaY), new Point(this.MouseDownDisplayArea.Right + deltaX,
      this.MouseDownDisplayArea.Bottom + deltaY));
01222
01223                     SetDisplayAreaNowInternal(target);
01224                     this.Cursor = new Cursor(StandardCursorType.SizeAll);
01225
01226                 }
01227                 else if (PointerEventHandlersType == PointerEventHandlers.Highlight ||
      (PointerEventHandlersType == PointerEventHandlers.PanHighlight && CurrentMouseOperation ==
      CurrentMouseOperations.Highlight))
01228                 {
01229
01230                     Point point = e.GetPosition(this);
01231
01232                     PointF pagePoint = new PointF((float)(point.X / this.Bounds.Width *
      DisplayArea.Width + DisplayArea.Left), (float)(point.Y / this.Bounds.Height * DisplayArea.Height +
      DisplayArea.Top));
01233
01234                     MuPDFStructuredTextAddress?  address =
      StructuredTextPage?.GetClosestHitAddress(pagePoint, false);
01235
01236                     this.Selection = new MuPDFStructuredTextAddressSpan(this.Selection.Start,
      address);
01237
01238                     if (address != null)
01239                     {
01240                         this.Cursor = new Cursor(StandardCursorType.Ibeam);
01241                     }
01242                     else
01243                     {
01244                         this.Cursor = new Cursor(StandardCursorType.Arrow);
01245                     }
01246                 }
01247             }
01248             else
01249             {
01250                 if (PointerEventHandlersType == PointerEventHandlers.Highlight ||
      PointerEventHandlersType == PointerEventHandlers.PanHighlight)
01251                 {
01252                     Point point = e.GetPosition(this);
01253
01254                     PointF pagePoint = new PointF((float)(point.X / this.Bounds.Width *
      DisplayArea.Width + DisplayArea.Left), (float)(point.Y / this.Bounds.Height * DisplayArea.Height +
      DisplayArea.Top));
01255
01256                     MuPDFStructuredTextAddress?  address =
      StructuredTextPage?.GetHitAddress(pagePoint, false);
01257
01258                     if (address != null)
01259                     {
01260                         this.Cursor = new Cursor(StandardCursorType.Ibeam);
01261                     }
01262                     else
01263                     {
01264                         this.Cursor = new Cursor(StandardCursorType.Arrow);
01265                     }
01266                 }
01267                 else
01268                 {
01269                     this.Cursor = new Cursor(StandardCursorType.Arrow);
01270                 }
01271             }
01272         }
01273
01274 /// <summary>
```

```
01275 /// Default handler for the PointerWheelChanged event (zoom in/out).
01276 /// </summary>
01277 /// <param name="sender"></param>
01278 /// <param name="e"></param>
01279         private void ControlPointerWheelChanged(object sender, PointerWheelEventArgs e)
01280         {
01281             if (ZoomEnabled)
01282             {
01283                 ZoomStep(e.Delta.Y, e.GetPosition(this));
01284             }
01285         }
01286 
01287 /// <summary>
01288 /// Compute the current value of the <see cref="Zoom"/> property.
01289 /// </summary>
01290         private void ComputeZoom()
01291         {
01292             //Take into account DPI scaling.
01293             double scale = (VisualRoot as ILayoutRoot).LayoutScaling;
01294             SetAndRaise(ZoomProperty, ref _Zoom, this.Bounds.Width / DisplayArea.Width * 72 / 96 *
      scale);
01295         }
01296 
01297 /// <summary>
01298 /// Draw the rendered document.
01299 /// </summary>
01300 /// <param name="context">The drawing context on which to draw.</param>
01301         public override void Render(DrawingContext context)
01302         {
01303             //Take into account DPI scaling.
01304             double scale = (VisualRoot as ILayoutRoot)?.LayoutScaling ??  1;
01305 
01306             context.FillRectangle(Background, this.Bounds);
01307 
01308             //Page boundaries (used to draw the page background).
01309             double minX = Math.Max(PageSize.Left, DisplayArea.Left);
01310             double maxX = Math.Min(PageSize.Right, DisplayArea.Right);
01311             double minY = Math.Max(PageSize.Top, DisplayArea.Top);
01312             double maxY = Math.Min(PageSize.Bottom, DisplayArea.Bottom);
01313 
01314 
01315             if (IsViewerInitialized)
01316             {
01317                 bool renderedDynamic = false;
01318 
01319                 //Check if someone is holding the mutex without blocking.
01320                 if (RenderMutex.WaitOne(0))
01321                 {
01322                     //Check if the DynamicBitmaps are ready
01323                     if (AreDynamicBitmapsReady)
01324                     {
01325                         //Page background
01326                         context.FillRectangle(PageBackground, new Rect(new Point((minX -
      DisplayArea.Left) / DisplayArea.Width * this.Bounds.Width, (minY - DisplayArea.Top) /
      DisplayArea.Height * this.Bounds.Height), new Point((maxX - DisplayArea.Left) / DisplayArea.Width *
      this.Bounds.Width, (maxY - DisplayArea.Top) / DisplayArea.Height * this.Bounds.Height)));
01327 
01328                         //Draw the DynamicBitmaps.
01329                         for (int i = 0; i < DynamicImagesBounds.Length; i++)
01330                         {
01331                             context.DrawImage(DynamicBitmaps[i], new Rect(new Point(0, 0),
      DynamicBitmaps[i].PixelSize.ToSize(1)), new Rect(DynamicImagesBounds[i].X0 / scale,
      DynamicImagesBounds[i].Y0 / scale, DynamicImagesBounds[i].Width / scale, DynamicImagesBounds[i].Height
      / scale));
01332                         }
01333 
01334                         //Signal that we don't need to draw the static image.
01335                         renderedDynamic = true;
01336                     }
01337 
01338                     //Release the mutex.
01339                     RenderMutex.ReleaseMutex();
01340                 }
01341 
01342                 //If the DynamicBitmaps have not been drawn, we fall back to drawing the static image
      (which will probably be ugly and pixelated, but better than nothing).
01343                 if (!renderedDynamic)
01344                 {
01345                     //Page background
01346                     context.FillRectangle(PageBackground, new Rect(new Point((minX - DisplayArea.Left)
      / DisplayArea.Width * this.Bounds.Width, (minY - DisplayArea.Top) / DisplayArea.Height *
      this.Bounds.Height), new Point((maxX - DisplayArea.Left) / DisplayArea.Width * this.Bounds.Width,
      (maxY - DisplayArea.Top) / DisplayArea.Height * this.Bounds.Height)));
01347 
01348                     //Top left corner of the DisplayArea in FixedCanvasBitmap coordinates.
01349                     Point topLeft = new Point((DisplayArea.X - FixedArea.X0) / FixedArea.Width *
      FixedCanvasBitmap.PixelSize.Width, (DisplayArea.Y - FixedArea.Y0) / FixedArea.Height *
```

```
    FixedCanvasBitmap.PixelSize.Height);
01350
01351                     //Size of the DisplayArea in FixedCanvasBitmap coordinates.
01352                     Avalonia.Size size = new Avalonia.Size(DisplayArea.Width / FixedArea.Width *
    FixedCanvasBitmap.PixelSize.Width, DisplayArea.Height / FixedArea.Height *
    FixedCanvasBitmap.PixelSize.Height);
01353
01354                     //Draw the FixedCanvasBitmap
01355                     context.DrawImage(FixedCanvasBitmap, new Rect(topLeft, size), new Rect(0, 0,
    this.Bounds.Width, this.Bounds.Height));
01356
01357                     //Draw the icon signaling that the DynamicBitmaps are still being rendered.
01358                     RefreshingGeometry.Transform = new TranslateTransform(this.Bounds.Width - 38, 32);
01359                     context.DrawGeometry(new SolidColorBrush(Color.FromRgb(119, 170, 221)), null,
    RefreshingGeometry);
01360                 }
01361
01362                 //Draw the highlight quads
01363                 if (this.HighlightQuads != null && this.HighlightQuads.Count > 0)
01364                 {
01365                     PathGeometry highlightGeometry = new PathGeometry() { FillRule = FillRule.NonZero
    };
01366
01367                     for (int i = 0; i < this.HighlightQuads.Count; i++)
01368                     {
01369                         Point ll = new Point((this.HighlightQuads[i].LowerLeft.X -
    this.DisplayArea.Left) * this.Bounds.Width / this.DisplayArea.Width,
    (this.HighlightQuads[i].LowerLeft.Y - this.DisplayArea.Top) * this.Bounds.Height /
    this.DisplayArea.Height);
01370                         Point ul = new Point((this.HighlightQuads[i].UpperLeft.X -
    this.DisplayArea.Left) * this.Bounds.Width / this.DisplayArea.Width,
    (this.HighlightQuads[i].UpperLeft.Y - this.DisplayArea.Top) * this.Bounds.Height /
    this.DisplayArea.Height);
01371                         Point ur = new Point((this.HighlightQuads[i].UpperRight.X -
    this.DisplayArea.Left) * this.Bounds.Width / this.DisplayArea.Width,
    (this.HighlightQuads[i].UpperRight.Y - this.DisplayArea.Top) * this.Bounds.Height /
    this.DisplayArea.Height);
01372                         Point lr = new Point((this.HighlightQuads[i].LowerRight.X -
    this.DisplayArea.Left) * this.Bounds.Width / this.DisplayArea.Width,
    (this.HighlightQuads[i].LowerRight.Y - this.DisplayArea.Top) * this.Bounds.Height /
    this.DisplayArea.Height);
01373
01374                         PathFigure quad = new PathFigure() { StartPoint = ll, IsClosed = true,
    IsFilled = true };
01375                         quad.Segments.Add(new LineSegment() { Point = ul });
01376                         quad.Segments.Add(new LineSegment() { Point = ur });
01377                         quad.Segments.Add(new LineSegment() { Point = lr });
01378
01379                         highlightGeometry.Figures.Add(quad);
01380                     }
01381
01382                     context.DrawGeometry(this.HighlightBrush, null, highlightGeometry);
01383                 }
01384
01385                 //Draw the selection quads
01386                 if (this.SelectionQuads != null && this.SelectionQuads.Count > 0)
01387                 {
01388                     PathGeometry selectionGeometry = new PathGeometry() { FillRule = FillRule.NonZero
    };
01389
01390                     for (int i = 0; i < this.SelectionQuads.Count; i++)
01391                     {
01392                         Point ll = new Point((this.SelectionQuads[i].LowerLeft.X -
    this.DisplayArea.Left) * this.Bounds.Width / this.DisplayArea.Width,
    (this.SelectionQuads[i].LowerLeft.Y - this.DisplayArea.Top) * this.Bounds.Height /
    this.DisplayArea.Height);
01393                         Point ul = new Point((this.SelectionQuads[i].UpperLeft.X -
    this.DisplayArea.Left) * this.Bounds.Width / this.DisplayArea.Width,
    (this.SelectionQuads[i].UpperLeft.Y - this.DisplayArea.Top) * this.Bounds.Height /
    this.DisplayArea.Height);
01394                         Point ur = new Point((this.SelectionQuads[i].UpperRight.X -
    this.DisplayArea.Left) * this.Bounds.Width / this.DisplayArea.Width,
    (this.SelectionQuads[i].UpperRight.Y - this.DisplayArea.Top) * this.Bounds.Height /
    this.DisplayArea.Height);
01395                         Point lr = new Point((this.SelectionQuads[i].LowerRight.X -
    this.DisplayArea.Left) * this.Bounds.Width / this.DisplayArea.Width,
    (this.SelectionQuads[i].LowerRight.Y - this.DisplayArea.Top) * this.Bounds.Height /
    this.DisplayArea.Height);
01396
01397                         PathFigure quad = new PathFigure() { StartPoint = ll, IsClosed = true,
    IsFilled = true };
01398                         quad.Segments.Add(new LineSegment() { Point = ul });
01399                         quad.Segments.Add(new LineSegment() { Point = ur });
01400                         quad.Segments.Add(new LineSegment() { Point = lr });
01401
01402                         selectionGeometry.Figures.Add(quad);
01403                     }
```

```
01404
01405                     context.DrawGeometry(this.SelectionBrush, null, selectionGeometry);
01406                 }
01407             }
01408         }
01409     }
01410 }
```

8.2 PDFRenderer.Properties.cs

```
00001 /*
00002 MuPDFCore.MuPDFRenderer - A control to display documents in Avalonia using MuPDFCore.
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 Avalonia;
00019 using Avalonia.Controls;
00020 using Avalonia.Layout;
00021 using Avalonia.Media;
00022 using System;
00023 using System.Collections.Generic;
00024
00025 namespace MuPDFCore.MuPDFRenderer
00026 {
00027     public partial class PDFRenderer :  Control
00028     {
00029 /// <summary>
00030 /// Defines the <see cref="RenderThreadCount"/> property.
00031 /// </summary>
00032         public static readonly DirectProperty<PDFRenderer, int> RenderThreadCountProperty =
      AvaloniaProperty.RegisterDirect<PDFRenderer, int>(nameof(RenderThreadCount), o =>
      o.RenderThreadCount);
00033 /// <summary>
00034 /// Backing field for the <see cref="RenderThreadCount"/> property.
00035 /// </summary>
00036         private int _RenderThreadCount;
00037 /// <summary>
00038 /// Exposes the number of threads that the current instance is using to render the document.
      Read-only.
00039 /// </summary>
00040         public int RenderThreadCount
00041         {
00042             get
00043             {
00044                 return _RenderThreadCount;
00045             }
00046
00047             private set
00048             {
00049                 SetAndRaise(RenderThreadCountProperty, ref _RenderThreadCount, value);
00050             }
00051         }
00052
00053 /// <summary>
00054 /// Defines the <see cref="PageNumber"/> property.
00055 /// </summary>
00056         public static readonly DirectProperty<PDFRenderer, int> PageNumberProperty =
      AvaloniaProperty.RegisterDirect<PDFRenderer, int>(nameof(PageNumber), o => o.PageNumber);
00057 /// <summary>
00058 /// Backing field for the <see cref="PageNumber"/> property.
00059 /// </summary>
00060         private int _PageNumber;
00061 /// <summary>
00062 /// Exposes the number of the page that the current instance is rendering.  Read-only.
00063 /// </summary>
00064         public int PageNumber
00065         {
00066             get
00067             {
00068                 return _PageNumber;
00069             }
```

```
00070
00071             private set
00072             {
00073                 SetAndRaise(PageNumberProperty, ref _PageNumber, value);
00074             }
00075         }
00076
00077 /// <summary>
00078 /// Defines the <see cref="IsViewerInitialized"/> property.
00079 /// </summary>
00080         public static readonly DirectProperty<PDFRenderer, bool> IsViewerInitializedProperty =
      AvaloniaProperty.RegisterDirect<PDFRenderer, bool>(nameof(IsViewerInitialized), o =>
      o.IsViewerInitialized);
00081 /// <summary>
00082 /// Backing field for the <see cref="IsViewerInitialized"/> property.
00083 /// </summary>
00084         private bool _IsViewerInitialized = false;
00085 /// <summary>
00086 /// Whether the current instance has been initialised with a document to render or not.  Read-only.
00087 /// </summary>
00088         public bool IsViewerInitialized
00089         {
00090             get
00091             {
00092                 return _IsViewerInitialized;
00093             }
00094
00095             private set
00096             {
00097                 SetAndRaise(IsViewerInitializedProperty, ref _IsViewerInitialized, value);
00098             }
00099         }
00100
00101 /// <summary>
00102 /// Defines the <see cref="PageSize"/> property.
00103 /// </summary>
00104         public static readonly DirectProperty<PDFRenderer, Rect> PageSizeProperty =
      AvaloniaProperty.RegisterDirect<PDFRenderer, Rect>(nameof(PageSize), o => o.PageSize);
00105 /// <summary>
00106 /// Backing field for the <see cref="PageSize"/> property.
00107 /// </summary>
00108         private Rect _PageSize;
00109 /// <summary>
00110 /// Exposes the size of the page that is drawn by the current instance (in page units).
00111 /// </summary>
00112         public Rect PageSize
00113         {
00114             get
00115             {
00116                 return _PageSize;
00117             }
00118
00119             private set
00120             {
00121                 SetAndRaise(PageSizeProperty, ref _PageSize, value);
00122             }
00123         }
00124
00125 /// <summary>
00126 /// Defines the <see cref="DisplayArea"/> property.
00127 /// </summary>
00128         public static readonly StyledProperty<Rect> DisplayAreaProperty =
      AvaloniaProperty.Register<PDFRenderer, Rect>(nameof(DisplayArea));
00129 /// <summary>
00130 /// The region of the page (in page units) that is currently displayed by the current instance.  This
      always has the same aspect ratio of the bounds of this control.
00131 /// When this is set, the value is sanitised so that the smallest rectangle with the correct aspect
      ratio containing the requested value is chosen.
00132 /// </summary>
00133         public Rect DisplayArea
00134         {
00135             get
00136             {
00137                 return GetValue(DisplayAreaProperty);
00138             }
00139
00140             set
00141             {
00142                 double widthRatio = value.Width / (this.Bounds.Width);
00143                 double heightRatio = value.Height / (this.Bounds.Height);
00144
00145                 double containingWidth = Math.Max(widthRatio, heightRatio) * this.Bounds.Width;
00146                 double containingHeight = Math.Max(widthRatio, heightRatio) * this.Bounds.Height;
00147
00148                 double deltaW = (containingWidth - value.Width) * 0.5;
00149                 double deltaH = (containingHeight - value.Height) * 0.5;
00150
```

```
00151                 Rect newDispArea = new Rect(new Point(value.X - deltaW, value.Y - deltaH), new
      Point(value.Right + deltaW, value.Bottom + deltaH));
00152
00153                 SetValue(DisplayAreaProperty, newDispArea);
00154             }
00155         }
00156
00157 /// <summary>
00158 /// Defines the <see cref="ZoomIncrement"/> property.
00159 /// </summary>
00160         public static readonly StyledProperty<double> ZoomIncrementProperty =
      AvaloniaProperty.Register<PDFRenderer, double>(nameof(ZoomIncrement), Math.Pow(2, 1.0 / 3.0),
      defaultBindingMode:  Avalonia.Data.BindingMode.TwoWay);
00161 /// <summary>
00162 /// Determines by how much the scale will be increased/decreased by the <see cref="ZoomStep(double,
      Point?)"/> method.  Set this to a value smaller than 1 to invert the zoom in/out direction.
00163 /// </summary>
00164         public double ZoomIncrement
00165         {
00166             get { return GetValue(ZoomIncrementProperty); }
00167
00168             set
00169             {
00170                 if (value <= 0)
00171                 {
00172                     throw new ArgumentOutOfRangeException(nameof(ZoomIncrement), value, "The
      ZoomIncrement must be greater than 0!");
00173                 }
00174
00175                 SetValue(ZoomIncrementProperty, value);
00176             }
00177         }
00178
00179 /// <summary>
00180 /// Defines the <see cref="Background"/> property.
00181 /// </summary>
00182         public static readonly StyledProperty<IBrush> BackgroundProperty =
      AvaloniaProperty.Register<PDFRenderer, IBrush>(nameof(Background));
00183 /// <summary>
00184 /// The background colour of the control.
00185 /// </summary>
00186         public IBrush Background
00187         {
00188             get { return GetValue(BackgroundProperty); }
00189             set { SetValue(BackgroundProperty, value); }
00190         }
00191
00192 /// <summary>
00193 /// Defines the <see cref="PageBackground"/> property.
00194 /// </summary>
00195         public static readonly StyledProperty<IBrush> PageBackgroundProperty =
      AvaloniaProperty.Register<PDFRenderer, IBrush>(nameof(PageBackground));
00196 /// <summary>
00197 /// The background colour to use for the page drawn by the control.
00198 /// </summary>
00199         public IBrush PageBackground
00200         {
00201             get { return GetValue(PageBackgroundProperty); }
00202             set { SetValue(PageBackgroundProperty, value); }
00203         }
00204
00205 /// <summary>
00206 /// Defines the <see cref="Zoom"/> property.
00207 /// </summary>
00208         public static readonly DirectProperty<PDFRenderer, double> ZoomProperty =
      AvaloniaProperty.RegisterDirect<PDFRenderer, double>(nameof(Zoom), o => o.Zoom, (o, v) => o.Zoom = v,
      defaultBindingMode:  Avalonia.Data.BindingMode.TwoWay);
00209 /// <summary>
00210 /// Backing field for the <see cref="Zoom"/> property.
00211 /// </summary>
00212         private double _Zoom;
00213 /// <summary>
00214 /// The current zoom level.  Setting this will change the <see cref="DisplayArea"/> appropriately,
      zooming around the center of the <see cref="DisplayArea"/>.
00215 /// </summary>
00216         public double Zoom
00217         {
00218             get
00219             {
00220                 return _Zoom;
00221             }
00222
00223             set
00224             {
00225                 double actualZoom = value / _Zoom;
00226
00227                 double currZoomX = FixedArea.Width / DisplayArea.Width * actualZoom;
```

```
00228                 double currZoomY = FixedArea.Height / DisplayArea.Height * actualZoom;
00229
00230                 double currWidth = FixedArea.Width / currZoomX;
00231                 double currHeight = FixedArea.Height / currZoomY;
00232
00233
00234                 Point pos = new Point(this.Bounds.Width * 0.5, this.Bounds.Height * 0.5);
00235
00236                 double deltaW = currWidth - DisplayArea.Width;
00237                 double deltaH = currHeight - DisplayArea.Height;
00238
00239                 SetValue(DisplayAreaProperty, new Rect(new Point(DisplayArea.X - deltaW * pos.X /
    this.Bounds.Width, DisplayArea.Y - deltaH * pos.Y / this.Bounds.Height), new Point(DisplayArea.Right +
    deltaW * (1 - pos.X / this.Bounds.Width), DisplayArea.Bottom + deltaH * (1 - pos.Y /
    this.Bounds.Height))));
00240             }
00241         }
00242
00243 /// <summary>
00244 /// Identifies the action to perform on pointer events.
00245 /// </summary>
00246         public enum PointerEventHandlers
00247         {
00248 /// <summary>
00249 /// Pointer events will be used to pan around the page.
00250 /// </summary>
00251             Pan,
00252
00253 /// <summary>
00254 /// Pointer events will be used to highlight text.
00255 /// </summary>
00256             Highlight,
00257
00258 /// <summary>
00259 /// Pointer events will be used to pan around the page or to highlight text, depending on where they
    start.
00260 /// </summary>
00261             PanHighlight,
00262
00263 /// <summary>
00264 /// Pointer events will be ignored.  If you use this value, you will have to implement your own way to
    pan around the document by changing the <see cref="DisplayArea"/> or to select text.
00265 /// </summary>
00266             Custom
00267         }
00268
00269 /// <summary>
00270 /// Defines the <see cref="PointerEventHandlersType"/> property.
00271 /// </summary>
00272         public static readonly StyledProperty<PointerEventHandlers> PointerEventHandlerTypeProperty =
    AvaloniaProperty.Register<PDFRenderer, PointerEventHandlers>(nameof(PointerEventHandlersType),
    PointerEventHandlers.PanHighlight);
00273 /// <summary>
00274 /// Whether the default handlers for pointer events (which are used for panning around the page)
    should be enabled.  If this is false, you will have to implement your own way to pan around the
    document by changing the <see cref="DisplayArea"/>.
00275 /// </summary>
00276         public PointerEventHandlers PointerEventHandlersType
00277         {
00278             get { return GetValue(PointerEventHandlerTypeProperty); }
00279             set { SetValue(PointerEventHandlerTypeProperty, value); }
00280         }
00281
00282 /// <summary>
00283 /// Defines the <see cref="ZoomEnabled"/> property.
00284 /// </summary>
00285         public static readonly StyledProperty<bool> ZoomEnabledProperty =
    AvaloniaProperty.Register<PDFRenderer, bool>(nameof(ZoomEnabled), true);
00286 /// <summary>
00287 /// Whether the default handlers for pointer wheel events (which are used for zooming in/out) should
    be enabled.  If this is false, you will have to implement your own way to zoom by changing the <see
    cref="DisplayArea"/>.
00288 /// </summary>
00289         public bool ZoomEnabled
00290         {
00291             get { return GetValue(ZoomEnabledProperty); }
00292             set { SetValue(ZoomEnabledProperty, value); }
00293         }
00294
00295 /// <summary>
00296 /// Defines the <see cref="Selection"/> property.
00297 /// </summary>
00298         public static readonly StyledProperty<MuPDFStructuredTextAddressSpan> SelectionProperty =
    AvaloniaProperty.Register<PDFRenderer, MuPDFStructuredTextAddressSpan>(nameof(Selection), null);
00299 /// <summary>
00300 /// The start and end of the currently selected text.
00301 /// </summary>
```

```
00302         public MuPDFStructuredTextAddressSpan Selection
00303         {
00304             get { return GetValue(SelectionProperty); }
00305             set { SetValue(SelectionProperty, value); }
00306         }
00307
00308 /// <summary>
00309 /// Defines the <see cref="SelectionBrush"/> property.
00310 /// </summary>
00311         public static readonly StyledProperty<IBrush> SelectionBrushProperty =
      AvaloniaProperty.Register<PDFRenderer, IBrush>(nameof(SelectionBrush), new
      SolidColorBrush(Color.FromArgb(96, 86, 180, 233)));
00312 /// <summary>
00313 /// The colour used to highlight the <see cref="Selection"/>.
00314 /// </summary>
00315         public IBrush SelectionBrush
00316         {
00317             get { return GetValue(SelectionBrushProperty); }
00318             set { SetValue(SelectionBrushProperty, value); }
00319         }
00320
00321 /// <summary>
00322 /// Defines the <see cref="HighlightedRegions"/> property.
00323 /// </summary>
00324         public static readonly StyledProperty<IEnumerable<MuPDFStructuredTextAddressSpan»
      HighlightedRegionsProperty = AvaloniaProperty.Register<PDFRenderer,
      IEnumerable<MuPDFStructuredTextAddressSpan»(nameof(HighlightedRegions), null);
00325 /// <summary>
00326 /// A collection of highlighted regions, e.g.  as a result of a text search.
00327 /// </summary>
00328         public IEnumerable<MuPDFStructuredTextAddressSpan> HighlightedRegions
00329         {
00330             get { return GetValue(HighlightedRegionsProperty); }
00331             set { SetValue(HighlightedRegionsProperty, value); }
00332         }
00333
00334 /// <summary>
00335 /// Defines the <see cref="HighlightBrush"/> property.
00336 /// </summary>
00337         public static readonly StyledProperty<IBrush> HighlightBrushProperty =
      AvaloniaProperty.Register<PDFRenderer, IBrush>(nameof(HighlightBrush), new
      SolidColorBrush(Color.FromArgb(96, 230, 159, 0)));
00338 /// <summary>
00339 /// The colour used to highlight the <see cref="HighlightedRegions"/>.
00340 /// </summary>
00341         public IBrush HighlightBrush
00342         {
00343             get { return GetValue(HighlightBrushProperty); }
00344             set { SetValue(HighlightBrushProperty, value); }
00345         }
00346     }
00347 }
```

8.3 RectTransition.cs

```
00001 /*
00002 MuPDFCore.MuPDFRenderer - A control to display documents in Avalonia using MuPDFCore.
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 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 namespace Avalonia.Animation
00019 {
00020 /// <summary>
00021 /// Transition class that handles <see cref="AvaloniaProperty"/> with <see cref="Rect"/> types.
00022 /// </summary>
00023     public class RectTransition :  InterpolatingTransitionBase<Rect>
00024     {
00025 /// <inheritdoc/>
00026         protected override Rect Interpolate(double f, Rect oldValue, Rect newValue)
00027         {
00028             return new Rect((newValue.X - oldValue.X) * f + oldValue.X,
```

```
00029                           (newValue.Y - oldValue.Y) * f + oldValue.Y,
00030                           (newValue.Width - oldValue.Width) * f + oldValue.Width,
00031                           (newValue.Height - oldValue.Height) * f + oldValue.Height);
00032         }
00033     }
00034 }
```

8.4 MuPDF.cs

```
00001 /*
00002 MuPDFCore - A set of multiplatform .NET Core bindings for MuPDF.
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 System;
00019 using System.IO;
00020 using System.IO.Pipes;
00021 using System.Runtime.InteropServices;
00022 using System.Text;
00023 using System.Threading.Tasks;
00024
00025 [assembly:  System.Runtime.CompilerServices.InternalsVisibleTo("Tests,
     PublicKey=0024000004800000940000000602000000240000525341310004000001000100d18d076ff369e4fb7295f51bbfedc5974e626236cec58
00026 namespace MuPDFCore
00027 {
00028 /// <summary>
00029 /// Exit codes returned by native methods describing various errors that can occur.
00030 /// </summary>
00031     public enum ExitCodes
00032     {
00033 /// <summary>
00034 /// An error occurred while creating the context object.
00035 /// </summary>
00036         ERR_CANNOT_CREATE_CONTEXT = 129,
00037
00038 /// <summary>
00039 /// An error occurred while registering the default document handlers with the context.
00040 /// </summary>
00041         ERR_CANNOT_REGISTER_HANDLERS = 130,
00042
00043 /// <summary>
00044 /// An error occurred while opening a file.
00045 /// </summary>
00046         ERR_CANNOT_OPEN_FILE = 131,
00047
00048 /// <summary>
00049 /// An error occurred while determining the total number of pages in the document.
00050 /// </summary>
00051         ERR_CANNOT_COUNT_PAGES = 132,
00052
00053 /// <summary>
00054 /// An error occurred while rendering the page.
00055 /// </summary>
00056         ERR_CANNOT_RENDER = 134,
00057
00058 /// <summary>
00059 /// An error occurred while opening the stream.
00060 /// </summary>
00061         ERR_CANNOT_OPEN_STREAM = 135,
00062
00063 /// <summary>
00064 /// An error occurred while loading the page.
00065 /// </summary>
00066         ERR_CANNOT_LOAD_PAGE = 136,
00067
00068 /// <summary>
00069 /// An error occurred while computing the page bounds.
00070 /// </summary>
00071         ERR_CANNOT_COMPUTE_BOUNDS = 137,
00072
00073 /// <summary>
```

```
00074 /// An error occurred while initialising the mutexes for the lock mechanism.
00075 /// </summary>
00076         ERR_CANNOT_INIT_MUTEX = 138,
00077
00078 /// <summary>
00079 /// An error occurred while cloning the context.
00080 /// </summary>
00081         ERR_CANNOT_CLONE_CONTEXT = 139,
00082
00083 /// <summary>
00084 /// An error occurred while saving the page to a raster image file.
00085 /// </summary>
00086         ERR_CANNOT_SAVE = 140,
00087
00088 /// <summary>
00089 /// An error occurred while creating the output buffer.
00090 /// </summary>
00091         ERR_CANNOT_CREATE_BUFFER = 141,
00092
00093 /// <summary>
00094 /// An error occurred while creating the document writer.
00095 /// </summary>
00096         ERR_CANNOT_CREATE_WRITER = 142,
00097
00098 /// <summary>
00099 /// An error occurred while finalising the document file.
00100 /// </summary>
00101         ERR_CANNOT_CLOSE_DOCUMENT = 143,
00102
00103 /// <summary>
00104 /// An error occurred while creating an empty structured text page.
00105 /// </summary>
00106         ERR_CANNOT_CREATE_PAGE = 144,
00107
00108 /// <summary>
00109 /// An error occurred while populating the structured text page
00110 /// </summary>
00111         ERR_CANNOT_POPULATE_PAGE = 145,
00112
00113 /// <summary>
00114 /// No error occurred.  All is well.
00115 /// </summary>
00116         EXIT_SUCCESS = 0
00117     }
00118
00119 /// <summary>
00120 /// File types supported in input by the library.
00121 /// </summary>
00122     public enum InputFileTypes
00123     {
00124 /// <summary>
00125 /// Portable Document Format.
00126 /// </summary>
00127         PDF = 0,
00128
00129 /// <summary>
00130 /// XML Paper Specification document.
00131 /// </summary>
00132         XPS = 1,
00133
00134 /// <summary>
00135 /// Comic book archive file (ZIP archive containing page scans).
00136 /// </summary>
00137         CBZ = 2,
00138
00139 /// <summary>
00140 /// Portable Network Graphics format.
00141 /// </summary>
00142         PNG = 3,
00143
00144 /// <summary>
00145 /// Joint Photographic Experts Group image.
00146 /// </summary>
00147         JPEG = 4,
00148
00149 /// <summary>
00150 /// Bitmap image.
00151 /// </summary>
00152         BMP = 5,
00153
00154 /// <summary>
00155 /// Graphics Interchange Format.
00156 /// </summary>
00157         GIF = 6,
00158
00159 /// <summary>
00160 /// Tagged Image File Format.
```

```
00161 /// </summary>
00162         TIFF = 7,
00163
00164 /// <summary>
00165 /// Portable aNyMap graphics format.
00166 /// </summary>
00167         PNM = 8,
00168
00169 /// <summary>
00170 /// Portable Arbitrary Map graphics format.
00171 /// </summary>
00172         PAM = 9,
00173
00174 /// <summary>
00175 /// Electronic PUBlication document.
00176 /// </summary>
00177         EPUB = 10,
00178
00179 /// <summary>
00180 /// FictionBook document.
00181 /// </summary>
00182         FB2 = 11,
00183
00184 /// <summary>
00185 /// Mobipocket e-book document.
00186 /// </summary>
00187         MOBI = 12,
00188
00189 /// <summary>
00190 /// HTML document.
00191 /// </summary>
00192         HTML = 13,
00193     }
00194
00195 /// <summary>
00196 /// Raster image file types supported in output by the library.
00197 /// </summary>
00198     public enum RasterOutputFileTypes
00199     {
00200 /// <summary>
00201 /// Portable aNyMap graphics format.
00202 /// </summary>
00203         PNM = 0,
00204
00205 /// <summary>
00206 /// Portable Arbitrary Map graphics format.
00207 /// </summary>
00208         PAM = 1,
00209
00210 /// <summary>
00211 /// Portable Network Graphics format.
00212 /// </summary>
00213         PNG = 2,
00214
00215 /// <summary>
00216 /// PhotoShop Document format.
00217 /// </summary>
00218         PSD = 3,
00219
00220 /// <summary>
00221 /// Joint Photographic Experts Group format, with quality level 90.
00222 /// </summary>
00223         JPEG = 4
00224     };
00225
00226 /// <summary>
00227 /// Document file types supported in output by the library.
00228 /// </summary>
00229     public enum DocumentOutputFileTypes
00230     {
00231 /// <summary>
00232 /// Portable Document Format.
00233 /// </summary>
00234         PDF = 0,
00235
00236 /// <summary>
00237 /// Scalable Vector Graphics.
00238 /// </summary>
00239         SVG = 1,
00240
00241 /// <summary>
00242 /// Comic book archive format.
00243 /// </summary>
00244         CBZ = 2
00245     };
00246
00247 /// <summary>
```

```
00248 /// Pixel formats supported by the library.
00249 /// </summary>
00250     public enum PixelFormats
00251     {
00252 /// <summary>
00253 /// 24bpp RGB format.
00254 /// </summary>
00255         RGB = 0,
00256 
00257 /// <summary>
00258 /// 32bpp RGBA format.
00259 /// </summary>
00260         RGBA = 1,
00261 
00262 /// <summary>
00263 /// 24bpp BGR format.
00264 /// </summary>
00265         BGR = 2,
00266 
00267 /// <summary>
00268 /// 32bpp BGRA format.
00269 /// </summary>
00270         BGRA = 3
00271     }
00272 
00273 /// <summary>
00274 /// Possible document encryption states.
00275 /// </summary>
00276     public enum EncryptionState
00277     {
00278 /// <summary>
00279 /// The document is not encrypted.
00280 /// </summary>
00281         Unencrypted = 0,
00282 
00283 /// <summary>
00284 /// The document is encrypted and a user password is necessary to render it.
00285 /// </summary>
00286         Encrypted = 1,
00287 
00288 /// <summary>
00289 /// The document is encrypted and the correct user password has been supplied.
00290 /// </summary>
00291         Unlocked = 2
00292     }
00293 
00294 /// <summary>
00295 /// Possible document restriction states.
00296 /// </summary>
00297     public enum RestrictionState
00298     {
00299 /// <summary>
00300 /// The document does not have any restrictions associated to it.
00301 /// </summary>
00302         Unrestricted = 0,
00303 
00304 /// <summary>
00305 /// Some restrictions apply to the document.  An owner password is required to remove these
    restrictions.
00306 /// </summary>
00307         Restricted = 1,
00308 
00309 /// <summary>
00310 /// The document had some restrictions and the correct owner password has been supplied.
00311 /// </summary>
00312         Unlocked = 2
00313     }
00314 
00315 /// <summary>
00316 /// Document restrictions.
00317 /// </summary>
00318     public enum DocumentRestrictions
00319     {
00320 /// <summary>
00321 /// No operation is restricted.
00322 /// </summary>
00323         None = 0,
00324 
00325 /// <summary>
00326 /// Printing the document is restricted.
00327 /// </summary>
00328         Print = 1,
00329 
00330 /// <summary>
00331 /// Copying the document is restricted.
00332 /// </summary>
00333         Copy = 2,
```

```
00334
00335 /// <summary>
00336 /// Editing the document is restricted.
00337 /// </summary>
00338         Edit = 4,
00339
00340 /// <summary>
00341 /// Annotating the document is restricted.
00342 /// </summary>
00343         Annotate = 8
00344     }
00345
00346 /// <summary>
00347 /// Password types.
00348 /// </summary>
00349     public enum PasswordTypes
00350     {
00351 /// <summary>
00352 /// No password.
00353 /// </summary>
00354         None = 0,
00355
00356 /// <summary>
00357 /// The password corresponds to the user password.
00358 /// </summary>
00359         User = 1,
00360
00361 /// <summary>
00362 /// The password corresponds to the owner password.
00363 /// </summary>
00364         Owner = 2
00365     }
00366
00367 /// <summary>
00368 /// A struct to hold information about the current rendering process and to abort rendering as needed.
00369 /// </summary>
00370     [StructLayout(LayoutKind.Sequential)]
00371     internal struct Cookie
00372     {
00373         public int abort;
00374         public int progress;
00375         public ulong progress_max;
00376         public int errors;
00377         public int incomplete;
00378     }
00379
00380 /// <summary>
00381 /// Holds a summary of the progress of the current rendering operation.
00382 /// </summary>
00383     public class RenderProgress
00384     {
00385 /// <summary>
00386 /// Holds the progress of a single thread.
00387 /// </summary>
00388         public struct ThreadRenderProgress
00389         {
00390 /// <summary>
00391 /// The current progress.
00392 /// </summary>
00393             public int Progress;
00394
00395 /// <summary>
00396 /// The maximum progress.  If this is 0, this value could not be determined (yet).
00397 /// </summary>
00398             public long MaxProgress;
00399
00400             internal ThreadRenderProgress(int progress, ulong maxProgress)
00401             {
00402                 this.Progress = progress;
00403                 this.MaxProgress = (long)maxProgress;
00404             }
00405         }
00406
00407 /// <summary>
00408 /// Contains the progress of all the threads used in rendering the document.
00409 /// </summary>
00410         public ThreadRenderProgress[] ThreadRenderProgresses { get; private set; }
00411
00412         internal RenderProgress(ThreadRenderProgress[] threadRenderProgresses)
00413         {
00414             ThreadRenderProgresses = threadRenderProgresses;
00415         }
00416     }
00417
00418 /// <summary>
00419 /// An <see cref="IDisposable"/> wrapper around an <see cref="IntPtr"/> that frees the allocated
      memory when it is disposed.
```

```
00420 /// </summary>
00421     public class DisposableIntPtr :  IDisposable
00422     {
00423 /// <summary>
00424 /// The pointer to the unmanaged memory.
00425 /// </summary>
00426         private readonly IntPtr InternalPointer;
00427 
00428 /// <summary>
00429 /// The number of bytes that have been allocated, for adding memory pressure.
00430 /// </summary>
00431         private readonly long BytesAllocated = -1;
00432 
00433 /// <summary>
00434 /// Create a new DisposableIntPtr.
00435 /// </summary>
00436 /// <param name="pointer">The pointer that should be freed upon disposing of this object.</param>
00437         public DisposableIntPtr(IntPtr pointer)
00438         {
00439             this.InternalPointer = pointer;
00440         }
00441 
00442 /// <summary>
00443 /// Create a new DisposableIntPtr, adding memory pressure to the GC to account for the allocation of
      unmanaged memory.
00444 /// </summary>
00445 /// <param name="pointer">The pointer that should be freed upon disposing of this object.</param>
00446 /// <param name="bytesAllocated">The number of bytes that have been allocated, for adding memory
      pressure.</param>
00447         public DisposableIntPtr(IntPtr pointer, long bytesAllocated)
00448         {
00449             this.InternalPointer = pointer;
00450             this.BytesAllocated = bytesAllocated;
00451 
00452             if (BytesAllocated > 0)
00453             {
00454                 GC.AddMemoryPressure(bytesAllocated);
00455             }
00456         }
00457 
00458         private bool disposedValue;
00459 
00460 ///<inheritdoc/>
00461         protected virtual void Dispose(bool disposing)
00462         {
00463             if (!disposedValue)
00464             {
00465                 Marshal.FreeHGlobal(InternalPointer);
00466 
00467                 if (BytesAllocated > 0)
00468                 {
00469                     GC.RemoveMemoryPressure(BytesAllocated);
00470                 }
00471 
00472                 disposedValue = true;
00473             }
00474         }
00475 
00476 ///<inheritdoc/>
00477         ~DisposableIntPtr()
00478         {
00479             Dispose(disposing:  false);
00480         }
00481 
00482 ///<inheritdoc/>
00483         public void Dispose()
00484         {
00485             Dispose(disposing:  true);
00486             GC.SuppressFinalize(this);
00487         }
00488     }
00489 
00490 /// <summary>
00491 /// The exception that is thrown when a MuPDF operation fails.
00492 /// </summary>
00493     public class MuPDFException :  Exception
00494     {
00495 /// <summary>
00496 /// The <see cref="ExitCodes"/> returned by the native function.
00497 /// </summary>
00498         public readonly ExitCodes ErrorCode;
00499 
00500         internal MuPDFException(string message, ExitCodes errorCode) :  base(message)
00501         {
00502             this.ErrorCode = errorCode;
00503         }
00504     }
```

```
00505
00506 /// <summary>
00507 /// The exception that is thrown when an attempt is made to render an encrypted document without
      supplying the required password.
00508 /// </summary>
00509     public class DocumentLockedException :  Exception
00510     {
00511         internal DocumentLockedException(string message) :  base(message) { }
00512     }
00513
00514 /// <summary>
00515 /// A class to simplify passing a string to the MuPDF C library with the correct encoding.
00516 /// </summary>
00517     internal class UTF8EncodedString :  IDisposable
00518     {
00519         private bool disposedValue;
00520
00521 /// <summary>
00522 /// The address of the bytes encoding the string in unmanaged memory.
00523 /// </summary>
00524         public IntPtr Address { get; }
00525
00526 /// <summary>
00527 /// Create a null-terminated, UTF-8 encoded string in unmanaged memory.
00528 /// </summary>
00529 /// <param name="text"></param>
00530         public UTF8EncodedString(string text)
00531         {
00532             byte[] data = System.Text.Encoding.UTF8.GetBytes(text);
00533
00534             IntPtr dataHolder = Marshal.AllocHGlobal(data.Length + 1);
00535             Marshal.Copy(data, 0, dataHolder, data.Length);
00536             Marshal.WriteByte(dataHolder, data.Length, 0);
00537
00538             this.Address = dataHolder;
00539         }
00540
00541         protected virtual void Dispose(bool disposing)
00542         {
00543             if (!disposedValue)
00544             {
00545                 Marshal.FreeHGlobal(Address);
00546                 disposedValue = true;
00547             }
00548         }
00549
00550         ~UTF8EncodedString()
00551         {
00552             Dispose(disposing:  false);
00553         }
00554
00555         public void Dispose()
00556         {
00557             Dispose(disposing:  true);
00558             GC.SuppressFinalize(this);
00559         }
00560     }
00561
00562 /// <summary>
00563 /// EventArgs for the <see cref="MuPDF.StandardOutputMessage"/> and <see
      cref="MuPDF.StandardErrorMessage"/> events.
00564 /// </summary>
00565     public class MessageEventArgs :  EventArgs
00566     {
00567 /// <summary>
00568 /// The message that has been logged.
00569 /// </summary>
00570         public string Message { get; }
00571
00572 /// <summary>
00573 /// Create a new <see cref="MessageEventArgs"/> instance.
00574 /// </summary>
00575 /// <param name="message">The message that has been logged.</param>
00576         public MessageEventArgs(string message)
00577         {
00578             this.Message = message;
00579         }
00580     }
00581
00582 /// <summary>
00583 /// Contains static methods to perform setup operations.
00584 /// </summary>
00585     public static class MuPDF
00586     {
00587         private static int StdOutFD = -1;
00588         private static int StdErrFD = -1;
00589
```

```
00590         private static TextWriter ConsoleOut;
00591         private static TextWriter ConsoleErr;
00592 
00593         private static ConsoleColor DefaultForeground;
00594         private static ConsoleColor DefaultBackground;
00595 
00596         private static string PipeName;
00597         private static bool CleanupRegistered = false;
00598         private static object CleanupLock = new object();
00599 
00600 /// <summary>
00601 /// This event is invoked when <see cref="RedirectOutput"/> has been called and the native MuPDF
      library writes to the standard output stream.
00602 /// </summary>
00603         public static event EventHandler<MessageEventArgs> StandardOutputMessage;
00604 
00605 /// <summary>
00606 /// This event is invoked when <see cref="RedirectOutput"/> has been called and the native MuPDF
      library writes to the standard error stream.
00607 /// </summary>
00608         public static event EventHandler<MessageEventArgs> StandardErrorMessage;
00609 
00610 /// <summary>
00611 /// Redirects output messages from the native MuPDF library to the <see cref="StandardOutputMessage"/>
      and <see cref="StandardErrorMessage"/> events.  Note that this has side-effects.
00612 /// </summary>
00613 /// <returns>A <see cref="Task"/> that finishes when the output streams have been
      redirected.</returns>
00614         public static async Task RedirectOutput()
00615         {
00616             if
      (System.Runtime.InteropServices.RuntimeInformation.IsOSPlatform(System.Runtime.InteropServices.OSPlatform.Windows))
00617             {
00618                 await RedirectOutputWindows();
00619             }
00620             else
00621             {
00622                 await RedirectOutputUnix();
00623             }
00624 
00625             if (!CleanupRegistered)
00626             {
00627                 AppDomain.CurrentDomain.ProcessExit += (s, e) =>
00628                 {
00629                     ResetOutput();
00630                 };
00631             }
00632         }
00633 
00634         const int UnixMaxPipeLength = 107;
00635 
00636         private static async Task RedirectOutputUnix()
00637         {
00638             if (StdOutFD < 0 && StdErrFD < 0)
00639             {
00640                 string tempPath = Path.GetTempPath();
00641 
00642                 string pipeName = "MuPDFCore-" + Guid.NewGuid().ToString();
00643 
00644                 pipeName = pipeName.Substring(0, Math.Min(pipeName.Length, UnixMaxPipeLength -
      tempPath.Length - 4));
00645                 pipeName = Path.Combine(tempPath, pipeName);
00646 
00647                 PipeName = pipeName;
00648 
00649                 Task redirectOutputTask = System.Threading.Tasks.Task.Run(() =>
00650                 {
00651                     NativeMethods.RedirectOutput(out StdOutFD, out StdErrFD, pipeName + "-out",
      pipeName + "-err");
00652                 });
00653 
00654                 // Start stdout pipe (this is actually a socket)
00655                 _ = Task.Run(() =>
00656                 {
00657                     using (NamedPipeClientStream client = new NamedPipeClientStream(pipeName +
      "-out"))
00658                     {
00659                         while (true)
00660                         {
00661                             try
00662                             {
00663                                 client.Connect(100);
00664                                 break;
00665                             }
00666                             catch { }
00667                         }
00668 
```

```
00669                         using (StreamReader reader = new StreamReader(client))
00670                         {
00671                             while (true)
00672                             {
00673                                 string message = reader.ReadLine();
00674 
00675                                 if (!string.IsNullOrEmpty(message))
00676                                 {
00677                                     StandardOutputMessage?.Invoke(null, new
      MessageEventArgs(message));
00678                                 }
00679                             }
00680                         }
00681                     }
00682 
00683                 });
00684 
00685                 // Start stderr pipe (this is actually a socket)
00686                 _ = Task.Run(() =>
00687                 {
00688                     using (NamedPipeClientStream client = new NamedPipeClientStream(pipeName +
      "-err"))
00689                     {
00690                         while (true)
00691                         {
00692                             try
00693                             {
00694                                 client.Connect(100);
00695                                 break;
00696                             }
00697                             catch { }
00698                         }
00699 
00700                         using (StreamReader reader = new StreamReader(client))
00701                         {
00702                             while (true)
00703                             {
00704                                 string message = reader.ReadLine();
00705 
00706                                 if (!string.IsNullOrEmpty(message))
00707                                 {
00708                                     StandardErrorMessage?.Invoke(null, new MessageEventArgs(message));
00709                                 }
00710                             }
00711                         }
00712                     }
00713 
00714                 });
00715 
00716                 await redirectOutputTask;
00717 
00718                 ConsoleOut = Console.Out;
00719                 ConsoleErr = Console.Error;
00720 
00721                 ConsoleColor fg = Console.ForegroundColor;
00722                 ConsoleColor bg = Console.BackgroundColor;
00723 
00724                 Console.ResetColor();
00725 
00726                 DefaultForeground = Console.ForegroundColor;
00727                 DefaultBackground = Console.BackgroundColor;
00728 
00729                 Console.ForegroundColor = fg;
00730                 Console.BackgroundColor = bg;
00731 
00732                 Console.SetOut(new FileDescriptorTextWriter(Console.Out.Encoding, StdOutFD));
00733                 Console.SetError(new FileDescriptorTextWriter(Console.Error.Encoding, StdErrFD));
00734             }
00735         }
00736 
00737         private static async Task RedirectOutputWindows()
00738         {
00739             if (StdOutFD < 0 && StdErrFD < 0)
00740             {
00741                 string pipeName = "MuPDFCore-" + Guid.NewGuid().ToString();
00742 
00743                 Task redirectOutputTask = System.Threading.Tasks.Task.Run(() =>
00744                 {
00745                     NativeMethods.RedirectOutput(out StdOutFD, out StdErrFD, "\\\\.\\pipe\\" +
      pipeName + "-out", "\\\\.\\pipe\\" + pipeName + "-err");
00746                 });
00747 
00748                 // Start stdout pipe
00749                 _ = Task.Run(() =>
00750                 {
00751                     using (NamedPipeClientStream client = new NamedPipeClientStream(pipeName +
      "-out"))
```

```
00752                     {
00753                         while (true)
00754                         {
00755                             try
00756                             {
00757                                 client.Connect(100);
00758                                 break;
00759                             }
00760                             catch { }
00761                         }
00762 
00763                         using (StreamReader reader = new StreamReader(client))
00764                         {
00765                             while (true)
00766                             {
00767                                 string message = reader.ReadLine();
00768 
00769                                 if (!string.IsNullOrEmpty(message))
00770                                 {
00771                                     StandardOutputMessage?.Invoke(null, new
      MessageEventArgs(message));
00772                                 }
00773                             }
00774                         }
00775                     }
00776 
00777                 });
00778 
00779                 // Start stderr pipe
00780                 _ = Task.Run(() =>
00781                 {
00782                     using (NamedPipeClientStream client = new NamedPipeClientStream(pipeName +
      "-err"))
00783                     {
00784                         while (true)
00785                         {
00786                             try
00787                             {
00788                                 client.Connect(100);
00789                                 break;
00790                             }
00791                             catch { }
00792                         }
00793 
00794                         using (StreamReader reader = new StreamReader(client))
00795                         {
00796                             while (true)
00797                             {
00798                                 string message = reader.ReadLine();
00799 
00800                                 if (!string.IsNullOrEmpty(message))
00801                                 {
00802                                     StandardErrorMessage?.Invoke(null, new MessageEventArgs(message));
00803                                 }
00804                             }
00805                         }
00806                     }
00807 
00808                 });
00809 
00810                 await redirectOutputTask;
00811 
00812                 ConsoleOut = Console.Out;
00813                 ConsoleErr = Console.Error;
00814 
00815                 ConsoleColor fg = Console.ForegroundColor;
00816                 ConsoleColor bg = Console.BackgroundColor;
00817 
00818                 Console.ResetColor();
00819 
00820                 DefaultForeground = Console.ForegroundColor;
00821                 DefaultBackground = Console.BackgroundColor;
00822 
00823                 Console.ForegroundColor = fg;
00824                 Console.BackgroundColor = bg;
00825 
00826                 Console.SetOut(new FileDescriptorTextWriter(Console.Out.Encoding, StdOutFD));
00827                 Console.SetError(new FileDescriptorTextWriter(Console.Error.Encoding, StdErrFD));
00828             }
00829         }
00830 
00831 /// <summary>
00832 /// Reset the default standard output and error streams for the native MuPDF library.
00833 /// </summary>
00834         public static void ResetOutput()
00835         {
00836             lock (CleanupLock)
```

```
00837             {
00838                 if (StdOutFD >= 0 && StdErrFD >= 0)
00839                 {
00840                     NativeMethods.ResetOutput(StdOutFD, StdErrFD);
00841 
00842                     Console.SetOut(ConsoleOut);
00843                     Console.SetError(ConsoleErr);
00844 
00845                     StdOutFD = -1;
00846                     StdErrFD = -1;
00847 
00848                     if
      (!System.Runtime.InteropServices.RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
00849                     {
00850                         File.Delete(PipeName + "-out");
00851                         File.Delete(PipeName + "-err");
00852                     }
00853                 }
00854             }
00855         }
00856 
00857         internal class FileDescriptorTextWriter :  TextWriter
00858         {
00859             public override Encoding Encoding { get; }
00860             private int FileDescriptor { get; }
00861 
00862             public FileDescriptorTextWriter(Encoding encoding, int fileDescriptor)
00863             {
00864                 this.Encoding = encoding;
00865                 this.FileDescriptor = fileDescriptor;
00866             }
00867 
00868             public override void Write(string value)
00869             {
00870                 StringBuilder sb = new StringBuilder();
00871 
00872                 if (Console.ForegroundColor != DefaultForeground || Console.BackgroundColor !=
      DefaultBackground)
00873                 {
00874                     sb.Append(" [");
00875                 }
00876 
00877                 if (Console.ForegroundColor != DefaultForeground)
00878                 {
00879                     switch (Console.ForegroundColor)
00880                     {
00881                         case ConsoleColor.Black:
00882                             sb.Append("30");
00883                             break;
00884                         case ConsoleColor.DarkRed:
00885                             sb.Append("31");
00886                             break;
00887                         case ConsoleColor.DarkGreen:
00888                             sb.Append("32");
00889                             break;
00890                         case ConsoleColor.DarkYellow:
00891                             sb.Append("33");
00892                             break;
00893                         case ConsoleColor.DarkBlue:
00894                             sb.Append("34");
00895                             break;
00896                         case ConsoleColor.DarkMagenta:
00897                             sb.Append("35");
00898                             break;
00899                         case ConsoleColor.DarkCyan:
00900                             sb.Append("36");
00901                             break;
00902                         case ConsoleColor.Gray:
00903                             sb.Append("37");
00904                             break;
00905                         case ConsoleColor.DarkGray:
00906                             sb.Append("90");
00907                             break;
00908                         case ConsoleColor.Red:
00909                             sb.Append("91");
00910                             break;
00911                         case ConsoleColor.Green:
00912                             sb.Append("92");
00913                             break;
00914                         case ConsoleColor.Yellow:
00915                             sb.Append("93");
00916                             break;
00917                         case ConsoleColor.Blue:
00918                             sb.Append("94");
00919                             break;
00920                         case ConsoleColor.Magenta:
00921                             sb.Append("95");
```

```
00922                           break;
00923                       case ConsoleColor.Cyan:
00924                           sb.Append("96");
00925                           break;
00926                       case ConsoleColor.White:
00927                           sb.Append("97");
00928                           break;
00929                   }
00930               }
00931
00932               if (Console.ForegroundColor != DefaultForeground && Console.BackgroundColor !=
      DefaultBackground)
00933               {
00934                   sb.Append(";");
00935               }
00936
00937               if (Console.BackgroundColor != DefaultBackground)
00938               {
00939                   switch (Console.BackgroundColor)
00940                   {
00941                       case ConsoleColor.Black:
00942                           sb.Append("40");
00943                           break;
00944                       case ConsoleColor.DarkRed:
00945                           sb.Append("41");
00946                           break;
00947                       case ConsoleColor.DarkGreen:
00948                           sb.Append("42");
00949                           break;
00950                       case ConsoleColor.DarkYellow:
00951                           sb.Append("43");
00952                           break;
00953                       case ConsoleColor.DarkBlue:
00954                           sb.Append("44");
00955                           break;
00956                       case ConsoleColor.DarkMagenta:
00957                           sb.Append("45");
00958                           break;
00959                       case ConsoleColor.DarkCyan:
00960                           sb.Append("46");
00961                           break;
00962                       case ConsoleColor.Gray:
00963                           sb.Append("47");
00964                           break;
00965                       case ConsoleColor.DarkGray:
00966                           sb.Append("100");
00967                           break;
00968                       case ConsoleColor.Red:
00969                           sb.Append("101");
00970                           break;
00971                       case ConsoleColor.Green:
00972                           sb.Append("102");
00973                           break;
00974                       case ConsoleColor.Yellow:
00975                           sb.Append("103");
00976                           break;
00977                       case ConsoleColor.Blue:
00978                           sb.Append("104");
00979                           break;
00980                       case ConsoleColor.Magenta:
00981                           sb.Append("105");
00982                           break;
00983                       case ConsoleColor.Cyan:
00984                           sb.Append("106");
00985                           break;
00986                       case ConsoleColor.White:
00987                           sb.Append("107");
00988                           break;
00989                   }
00990               }
00991
00992               if (Console.ForegroundColor != DefaultForeground || Console.BackgroundColor !=
      DefaultBackground)
00993               {
00994                   sb.Append("m");
00995               }
00996
00997               sb.Append(value);
00998
00999               if (Console.ForegroundColor != DefaultForeground || Console.BackgroundColor !=
      DefaultBackground)
01000               {
01001                   sb.Append(" [");
01002               }
01003
01004               if (Console.ForegroundColor != DefaultForeground)
01005               {
```

```
01006                     sb.Append("39");
01007                 }
01008 
01009                 if (Console.ForegroundColor != DefaultForeground && Console.BackgroundColor !=
      DefaultBackground)
01010                 {
01011                     sb.Append(";");
01012                 }
01013 
01014                 if (Console.BackgroundColor != DefaultBackground)
01015                 {
01016                     sb.Append("49");
01017                 }
01018 
01019                 if (Console.ForegroundColor != DefaultForeground || Console.BackgroundColor !=
      DefaultBackground)
01020                 {
01021                     sb.Append("m");
01022                 }
01023 
01024                 NativeMethods.WriteToFileDescriptor(FileDescriptor, sb.ToString(), sb.Length);
01025             }
01026 
01027             public override void Write(char value)
01028             {
01029                 Write(value.ToString());
01030             }
01031 
01032             public override void Write(char[] buffer)
01033             {
01034                 Write(new string(buffer));
01035             }
01036 
01037             public override void Write(char[] buffer, int index, int count)
01038             {
01039                 Write(new string(buffer, index, count));
01040             }
01041 
01042             public override void WriteLine(string value)
01043             {
01044                 Write(value);
01045                 WriteLine();
01046             }
01047         }
01048     }
01049 
01050 /// <summary>
01051 /// Native methods.
01052 /// </summary>
01053     internal static class NativeMethods
01054     {
01055 /// <summary>
01056 /// Create a MuPDF context object with the specified store size.
01057 /// </summary>
01058 /// <param name="store_size">Maximum size in bytes of the resource store.</param>
01059 /// <param name="out_ctx">A pointer to the native context object.</param>
01060 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
01061         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01062         internal static extern int CreateContext(ulong store_size, ref IntPtr out_ctx);
01063 
01064 /// <summary>
01065 /// Free a context and its global store.
01066 /// </summary>
01067 /// <param name="ctx">A pointer to the native context to free.</param>
01068 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
01069         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01070         internal static extern int DisposeContext(IntPtr ctx);
01071 
01072 /// <summary>
01073 /// Evict items from the store until the total size of the objects in the store is reduced to a given
      percentage of its current size.
01074 /// </summary>
01075 /// <param name="ctx">The context whose store should be shrunk.</param>
01076 /// <param name="perc">Fraction of current size to reduce the store to.</param>
01077 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
01078         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01079         internal static extern int ShrinkStore(IntPtr ctx, uint perc);
01080 
01081 /// <summary>
01082 /// Evict every item from the store.
01083 /// </summary>
01084 /// <param name="ctx">The context whose store should be emptied.</param>
01085 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
```

```
01086         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01087         internal static extern void EmptyStore(IntPtr ctx);
01088
01089 /// <summary>
01090 /// Get the current size of the store.
01091 /// </summary>
01092 /// <param name="ctx">The context whose store's size should be determined.</param>
01093 /// <returns>The current size in bytes of the store.</returns>
01094         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01095         internal static extern ulong GetCurrentStoreSize(IntPtr ctx);
01096
01097 /// <summary>
01098 /// Get the maximum size of the store.
01099 /// </summary>
01100 /// <param name="ctx">The context whose store's maximum size should be determined.</param>
01101 /// <returns>The maximum size in bytes of the store.</returns>
01102         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01103         internal static extern ulong GetMaxStoreSize(IntPtr ctx);
01104
01105 /// <summary>
01106 /// Set the current antialiasing levels.
01107 /// </summary>
01108 /// <param name="ctx">The context whose antialiasing levels should be set.</param>
01109 /// <param name="aa">The overall antialiasing level.  Ignored if &lt; 0.</param>
01110 /// <param name="graphics_aa">The graphics antialiasing level.  Ignored if &lt; 0.</param>
01111 /// <param name="text_aa">The text antialiasing level.  Ignored if &lt; 0.</param>
01112         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01113         internal static extern void SetAALevel(IntPtr ctx, int aa, int graphics_aa, int text_aa);
01114
01115 /// <summary>
01116 /// Get the current antialiasing levels.
01117 /// </summary>
01118 /// <param name="ctx">The context whose antialiasing levels should be retrieved.</param>
01119 /// <param name="out_aa">The overall antialiasing level.</param>
01120 /// <param name="out_graphics_aa">The graphics antialiasing level.</param>
01121 /// <param name="out_text_aa">The text antialiasing level.</param>
01122         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01123         internal static extern void GetAALevel(IntPtr ctx, out int out_aa, out int out_graphics_aa,
      out int out_text_aa);
01124
01125 /// <summary>
01126 /// Create a display list from a page.
01127 /// </summary>
01128 /// <param name="ctx">A pointer to the context used to create the document.</param>
01129 /// <param name="page">A pointer to the page that should be used to create the display list.</param>
01130 /// <param name="annotations">An integer indicating whether annotations should be included in the
      display list (1) or not (any other value).</param>
01131 /// <param name="out_display_list">A pointer to the newly-created display list.</param>
01132 /// <param name="out_x0">The left coordinate of the display list's bounds.</param>
01133 /// <param name="out_y0">The top coordinate of the display list's bounds.</param>
01134 /// <param name="out_x1">The right coordinate of the display list's bounds.</param>
01135 /// <param name="out_y1">The bottom coordinate of the display list's bounds.</param>
01136 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
01137         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01138         internal static extern int GetDisplayList(IntPtr ctx, IntPtr page, int annotations, ref IntPtr
      out_display_list, ref float out_x0, ref float out_y0, ref float out_x1, ref float out_y1);
01139
01140 /// <summary>
01141 /// Free a display list.
01142 /// </summary>
01143 /// <param name="ctx">The context that was used to create the display list.</param>
01144 /// <param name="list">The display list to dispose.</param>
01145 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
01146         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01147         internal static extern int DisposeDisplayList(IntPtr ctx, IntPtr list);
01148
01149 /// <summary>
01150 /// Create a new document from a stream.
01151 /// </summary>
01152 /// <param name="ctx">The context to which the document will belong.</param>
01153 /// <param name="data">A pointer to a byte array containing the data that makes up the
      document.</param>
01154 /// <param name="data_length">The length in bytes of the data that makes up the document.</param>
01155 /// <param name="file_type">The type (extension) of the document.</param>
01156 /// <param name="get_image_resolution">If this is not 0, try opening the stream as an image and return
      the actual resolution (in DPI) of the image.  Otherwise (or if trying to open the stream as an image
      fails), the returned resolution will be -1.</param>
01157 /// <param name="out_doc">The newly created document.</param>
01158 /// <param name="out_str">The newly created stream (so that it can be disposed later).</param>
01159 /// <param name="out_page_count">The number of pages in the document.</param>
01160 /// <param name="out_image_xres">If the document is an image file, the horizontal resolution of the
      image.</param>
01161 /// <param name="out_image_yres">If the document is an image file, the vertical resolution of the
      image.</param>
01162 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
```

```
      occurred.</returns>
01163         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01164         internal static extern int CreateDocumentFromStream(IntPtr ctx, IntPtr data, ulong
      data_length, string file_type, int get_image_resolution, ref IntPtr out_doc, ref IntPtr out_str, ref
      int out_page_count, ref float out_image_xres, ref float out_image_yres);
01165
01166 /// <summary>
01167 /// Create a new document from a file name.
01168 /// </summary>
01169 /// <param name="ctx">The context to which the document will belong.</param>
01170 /// <param name="file_name">The path of the file to open, UTF-8 encoded.</param>
01171 /// <param name="get_image_resolution">If this is not 0, try opening the file as an image and return
      the actual resolution (in DPI) of the image.  Otherwise (or if trying to open the file as an image
      fails), the returned resolution will be -1.</param>
01172 /// <param name="out_doc">The newly created document.</param>
01173 /// <param name="out_page_count">The number of pages in the document.</param>
01174 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
01175 /// <param name="out_image_xres">If the document is an image file, the horizontal resolution of the
      image.</param>
01176 /// <param name="out_image_yres">If the document is an image file, the vertical resolution of the
      image.</param>
01177         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01178         internal static extern int CreateDocumentFromFile(IntPtr ctx, IntPtr file_name, int
      get_image_resolution, ref IntPtr out_doc, ref int out_page_count, ref float out_image_xres, ref float
      out_image_yres);
01179
01180 /// <summary>
01181 /// Free a stream and its associated resources.
01182 /// </summary>
01183 /// <param name="ctx">The context that was used while creating the stream.</param>
01184 /// <param name="str">The stream to free.</param>
01185 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
01186         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01187         internal static extern int DisposeStream(IntPtr ctx, IntPtr str);
01188
01189 /// <summary>
01190 /// Free a document and its associated resources.
01191 /// </summary>
01192 /// <param name="ctx">The context that was used in creating the document.</param>
01193 /// <param name="doc">The document to free.</param>
01194 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
01195         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01196         internal static extern int DisposeDocument(IntPtr ctx, IntPtr doc);
01197
01198 /// <summary>
01199 /// Render (part of) a display list to an array of bytes starting at the specified pointer.
01200 /// </summary>
01201 /// <param name="ctx">A context to hold the exception stack and the cached resources.</param>
01202 /// <param name="list">The display list to render.</param>
01203 /// <param name="x0">The left coordinate in page units of the region of the display list that should
      be rendererd.</param>
01204 /// <param name="y0">The top coordinate in page units of the region of the display list that should be
      rendererd.</param>
01205 /// <param name="x1">The right coordinate in page units of the region of the display list that should
      be rendererd.</param>
01206 /// <param name="y1">The bottom coordinate in page units of the region of the display list that should
      be rendererd.</param>
01207 /// <param name="zoom">How much the specified region should be scaled when rendering.  This determines
      the size in pixels of the rendered image.</param>
01208 /// <param name="colorFormat">The pixel data format.</param>
01209 /// <param name="pixel_storage">A pointer indicating where the pixel bytes will be written.  There
      must be enough space available!</param>
01210 /// <param name="cookie">A pointer to a cookie object that can be used to track progress and/or abort
      rendering.  Can be null.</param>
01211 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
01212         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01213         internal static extern int RenderSubDisplayList(IntPtr ctx, IntPtr list, float x0, float y0,
      float x1, float y1, float zoom, int colorFormat, IntPtr pixel_storage, IntPtr cookie);
01214
01215 /// <summary>
01216 /// Load a page from a document.
01217 /// </summary>
01218 /// <param name="ctx">The context to which the document belongs.</param>
01219 /// <param name="doc">The document from which the page should be extracted.</param>
01220 /// <param name="page_number">The page number.</param>
01221 /// <param name="out_page">The newly extracted page.</param>
01222 /// <param name="out_x">The left coordinate of the page's bounds.</param>
01223 /// <param name="out_y">The top coordinate of the page's bounds.</param>
01224 /// <param name="out_w">The width of the page.</param>
01225 /// <param name="out_h">The height of the page.</param>
01226 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
01227         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
```

```
01228         internal static extern int LoadPage(IntPtr ctx, IntPtr doc, int page_number, ref IntPtr
      out_page, ref float out_x, ref float out_y, ref float out_w, ref float out_h);
01229
01230 /// <summary>
01231 /// Free a page and its associated resources.
01232 /// </summary>
01233 /// <param name="ctx">The context to which the document containing the page belongs.</param>
01234 /// <param name="page">The page to free.</param>
01235 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
01236         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01237         internal static extern int DisposePage(IntPtr ctx, IntPtr page);
01238
01239 /// <summary>
01240 /// Layout reflowable document types.
01241 /// </summary>
01242 /// <param name="ctx">The context to which the document belongs.</param>
01243 /// <param name="doc">The document to layout.</param>
01244 /// <param name="width">The page width.</param>
01245 /// <param name="height">The page height.</param>
01246 /// <param name="em">The default font size, in points.</param>
01247 /// <param name="out_page_count">The number of pages in the document, after the layout.</param>
01248 /// <returns>An integer detailing whether any errors occurred.</returns>
01249         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01250         internal static extern int LayoutDocument(IntPtr ctx, IntPtr doc, float width, float height,
      float em, out int out_page_count);
01251
01252 /// <summary>
01253 /// Create cloned contexts that can be used in multithreaded rendering.
01254 /// </summary>
01255 /// <param name="ctx">The original context to clone.</param>
01256 /// <param name="count">The number of cloned contexts to create.</param>
01257 /// <param name="out_contexts">An array of pointers to the cloned contexts.</param>
01258 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
01259         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01260         internal static extern int CloneContext(IntPtr ctx, int count, IntPtr out_contexts);
01261
01262 /// <summary>
01263 /// Save (part of) a display list to an image file in the specified format.
01264 /// </summary>
01265 /// <param name="ctx">A context to hold the exception stack and the cached resources.</param>
01266 /// <param name="list">The display list to render.</param>
01267 /// <param name="x0">The left coordinate in page units of the region of the display list that should
      be rendererd.</param>
01268 /// <param name="y0">The top coordinate in page units of the region of the display list that should be
      rendererd.</param>
01269 /// <param name="x1">The right coordinate in page units of the region of the display list that should
      be rendererd.</param>
01270 /// <param name="y1">The bottom coordinate in page units of the region of the display list that should
      be rendererd.</param>
01271 /// <param name="zoom">How much the specified region should be scaled when rendering.  This determines
      the size in pixels of the rendered image.</param>
01272 /// <param name="colorFormat">The pixel data format.</param>
01273 /// <param name="file_name">The path to the output file, UTF-8 encoded.</param>
01274 /// <param name="output_format">An integer equivalent to <see cref="RasterOutputFileTypes"/>
      specifying the output format.</param>
01275 /// <param name="quality">Quality level for the output format (where applicable).</param>
01276 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
01277         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01278         internal static extern int SaveImage(IntPtr ctx, IntPtr list, float x0, float y0, float x1,
      float y1, float zoom, int colorFormat, IntPtr file_name, int output_format, int quality);
01279
01280 /// <summary>
01281 /// Write (part of) a display list to an image buffer in the specified format.
01282 /// </summary>
01283 /// <param name="ctx">A context to hold the exception stack and the cached resources.</param>
01284 /// <param name="list">The display list to render.</param>
01285 /// <param name="x0">The left coordinate in page units of the region of the display list that should
      be rendererd.</param>
01286 /// <param name="y0">The top coordinate in page units of the region of the display list that should be
      rendererd.</param>
01287 /// <param name="x1">The right coordinate in page units of the region of the display list that should
      be rendererd.</param>
01288 /// <param name="y1">The bottom coordinate in page units of the region of the display list that should
      be rendererd.</param>
01289 /// <param name="zoom">How much the specified region should be scaled when rendering.  This determines
      the size in pixels of the rendered image.</param>
01290 /// <param name="colorFormat">The pixel data format.</param>
01291 /// <param name="output_format">An integer equivalent to <see cref="RasterOutputFileTypes"/>
      specifying the output format.</param>
01292 /// <param name="quality">Quality level for the output format (where applicable).</param>
01293 /// <param name="out_buffer">The address of the buffer on which the data has been written (only useful
      for disposing the buffer later).</param>
01294 /// <param name="out_data">The address of the byte array where the data has been actually
      written.</param>
```

```
01295 /// <param name="out_length">The length in bytes of the image data.</param>
01296 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
01297         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01298         internal static extern int WriteImage(IntPtr ctx, IntPtr list, float x0, float y0, float x1,
      float y1, float zoom, int colorFormat, int output_format, int quality, ref IntPtr out_buffer, ref
      IntPtr out_data, ref ulong out_length);
01299
01300 /// <summary>
01301 /// Free a native buffer and its associated resources.
01302 /// </summary>
01303 /// <param name="ctx">A context to hold the exception stack and the cached resources.</param>
01304 /// <param name="buf">The buffer to free.</param>
01305 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
01306         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01307         internal static extern int DisposeBuffer(IntPtr ctx, IntPtr buf);
01308
01309 /// <summary>
01310 /// Create a new document writer object.
01311 /// </summary>
01312 /// <param name="ctx">A context to hold the exception stack and the cached resources.</param>
01313 /// <param name="file_name">The name of file that will hold the writer's output, UTF-8
      encoded.</param>
01314 /// <param name="format">An integer equivalent to <see cref="DocumentOutputFileTypes"/> specifying the
      output format.</param>
01315 /// <param name="out_document_writer">A pointer to the new document writer object.</param>
01316 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
01317         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01318         internal static extern int CreateDocumentWriter(IntPtr ctx, IntPtr file_name, int format, ref
      IntPtr out_document_writer);
01319
01320 /// <summary>
01321 /// Render (part of) a display list as a page in the specified document writer.
01322 /// </summary>
01323 /// <param name="ctx">A context to hold the exception stack and the cached resources.</param>
01324 /// <param name="list">The display list to render.</param>
01325 /// <param name="x0">The left coordinate in page units of the region of the display list that should
      be rendererd.</param>
01326 /// <param name="y0">The top coordinate in page units of the region of the display list that should be
      rendererd.</param>
01327 /// <param name="x1">The right coordinate in page units of the region of the display list that should
      be rendererd.</param>
01328 /// <param name="y1">The bottom coordinate in page units of the region of the display list that should
      be rendererd.</param>
01329 /// <param name="zoom">How much the specified region should be scaled when rendering.  This will
      determine the final size of the page.</param>
01330 /// <param name="writ">The document writer on which the page should be written.</param>
01331 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
01332         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01333         internal static extern int WriteSubDisplayListAsPage(IntPtr ctx, IntPtr list, float x0, float
      y0, float x1, float y1, float zoom, IntPtr writ);
01334
01335 /// <summary>
01336 /// Finalise a document writer, closing the file and freeing all resources.
01337 /// </summary>
01338 /// <param name="ctx">The context that was used to create the document writer.</param>
01339 /// <param name="writ">The document writer to finalise.</param>
01340 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
01341         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01342         internal static extern int FinalizeDocumentWriter(IntPtr ctx, IntPtr writ);
01343
01344 /// <summary>
01345 /// Get the contents of a structured text character.
01346 /// </summary>
01347 /// <param name="character">The address of the character.</param>
01348 /// <param name="out_c">Unicode code point of the character.</param>
01349 /// <param name="out_color">An sRGB hex representation of the colour of the character.</param>
01350 /// <param name="out_origin_x">The x coordinate of the baseline origin of the character.</param>
01351 /// <param name="out_origin_y">The y coordinate of the baseline origin of the character.</param>
01352 /// <param name="out_size">The size in points of the character.</param>
01353 /// <param name="out_ll_x">The x coordinate of the lower left corner of the bounding quad.</param>
01354 /// <param name="out_ll_y">The y coordinate of the lower left corner of the bounding quad.</param>
01355 /// <param name="out_ul_x">The x coordinate of the upper left corner of the bounding quad.</param>
01356 /// <param name="out_ul_y">The y coordinate of the upper left corner of the bounding quad.</param>
01357 /// <param name="out_ur_x">The x coordinate of the upper right corner of the bounding quad.</param>
01358 /// <param name="out_ur_y">The y coordinate of the upper right corner of the bounding quad.</param>
01359 /// <param name="out_lr_x">The x coordinate of the lower right corner of the bounding quad.</param>
01360 /// <param name="out_lr_y">The y coordinate of the lower right corner of the bounding quad.</param>
01361 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
01362         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01363         internal static extern int GetStructuredTextChar(IntPtr character, ref int out_c, ref int
      out_color, ref float out_origin_x, ref float out_origin_y, ref float out_size, ref float out_ll_x, ref
```

```
      float out_ll_y, ref float out_ul_x, ref float out_ul_y, ref float out_ur_x, ref float out_ur_y, ref
      float out_lr_x, ref float out_lr_y);
01364
01365 /// <summary>
01366 /// Get an array of structured text characters from a structured text line.
01367 /// </summary>
01368 /// <param name="line">The structured text line from which the characters should be extracted.</param>
01369 /// <param name="out_chars">An array of pointers to the structured text characters.</param>
01370 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
01371         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01372         internal static extern int GetStructuredTextChars(IntPtr line, IntPtr out_chars);
01373
01374 /// <summary>
01375 /// Get the contents of a structured text line.
01376 /// </summary>
01377 /// <param name="line">The address of the line.</param>
01378 /// <param name="out_wmode">An integer equivalent to <see cref="MuPDFStructuredTextLine"/>
      representing the writing mode of the line.</param>
01379 /// <param name="out_x0">The left coordinate in page units of the bounding box of the line.</param>
01380 /// <param name="out_y0">The top coordinate in page units of the bounding box of the line.</param>
01381 /// <param name="out_x1">The right coordinate in page units of the bounding box of the line.</param>
01382 /// <param name="out_y1">The bottom coordinate in page units of the bounding box of the line.</param>
01383 /// <param name="out_x">The x component of the normalised direction of the baseline.</param>
01384 /// <param name="out_y">The y component of the normalised direction of the baseline.</param>
01385 /// <param name="out_char_count">The number of characters in the line.</param>
01386 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
01387         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01388         internal static extern int GetStructuredTextLine(IntPtr line, ref int out_wmode, ref float
      out_x0, ref float out_y0, ref float out_x1, ref float out_y1, ref float out_x, ref float out_y, ref
      int out_char_count);
01389
01390 /// <summary>
01391 /// Get an array of structured text lines from a structured text block.
01392 /// </summary>
01393 /// <param name="block">The structured text block from which the lines should be extracted.</param>
01394 /// <param name="out_lines">An array of pointers to the structured text lines.</param>
01395 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
01396         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01397         internal static extern int GetStructuredTextLines(IntPtr block, IntPtr out_lines);
01398
01399 /// <summary>
01400 /// Get the contents of a structured text block.
01401 /// </summary>
01402 /// <param name="block">The address of the block.</param>
01403 /// <param name="out_type">An integer equivalent to <see cref="MuPDFStructuredTextBlock.Types"/>
      representing the type of the block.</param>
01404 /// <param name="out_x0">The left coordinate in page units of the bounding box of the block.</param>
01405 /// <param name="out_y0">The top coordinate in page units of the bounding box of the block.</param>
01406 /// <param name="out_x1">The right coordinate in page units of the bounding box of the block.</param>
01407 /// <param name="out_y1">The bottom coordinate in page units of the bounding box of the block.</param>
01408 /// <param name="out_line_count">The number of lines in the block.</param>
01409 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
01410         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01411         internal static extern int GetStructuredTextBlock(IntPtr block, ref int out_type, ref float
      out_x0, ref float out_y0, ref float out_x1, ref float out_y1, ref int out_line_count);
01412
01413 /// <summary>
01414 /// Get an array of structured text blocks from a structured text page.
01415 /// </summary>
01416 /// <param name="page">The structured text page from which the blocks should be extracted.</param>
01417 /// <param name="out_blocks">An array of pointers to the structured text blocks.</param>
01418 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
01419         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01420         internal static extern int GetStructuredTextBlocks(IntPtr page, IntPtr out_blocks);
01421
01422 /// <summary>
01423 /// Get a structured text representation of a display list.
01424 /// </summary>
01425 /// <param name="ctx">A context to hold the exception stack and the cached resources.</param>
01426 /// <param name="list">The display list whose structured text representation is sought.</param>
01427 /// <param name="out_page">The address of the structured text page.</param>
01428 /// <param name="out_stext_block_count">The number of structured text blocks in the page.</param>
01429 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
01430         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01431         internal static extern int GetStructuredTextPage(IntPtr ctx, IntPtr list, ref IntPtr out_page,
      ref int out_stext_block_count);
01432
01433 /// <summary>
01434 /// Delegate defining a callback function that is invoked by the unmanaged MuPDF library to indicate
      OCR progress.
01435 /// </summary>
```

```
01436 /// <param name="progress">The current progress, ranging from 0 to 100.</param>
01437 /// <returns>This function should return 0 to indicate that the OCR process should continue, or 1 to
      indicate that it should be stopped.</returns>
01438         internal delegate int ProgressCallback(int progress);
01439
01440 /// <summary>
01441 /// Get a structured text representation of a display list, using the Tesseract OCR engine.
01442 /// </summary>
01443 /// <param name="ctx">A context to hold the exception stack and the cached resources.</param>
01444 /// <param name="list">The display list whose structured text representation is sought.</param>
01445 /// <param name="out_page">The address of the structured text page.</param>
01446 /// <param name="out_stext_block_count">The number of structured text blocks in the page.</param>
01447 /// <param name="zoom">How much the specified region should be scaled when rendering.  This determines
      the size in pixels of the image that is passed to Tesseract.</param>
01448 /// <param name="x0">The left coordinate in page units of the region of the display list that should
      be analysed.</param>
01449 /// <param name="y0">The top coordinate in page units of the region of the display list that should be
      analysed.</param>
01450 /// <param name="x1">The right coordinate in page units of the region of the display list that should
      be analysed.</param>
01451 /// <param name="y1">The bottom coordinate in page units of the region of the display list that should
      be analysed.</param>
01452 /// <param name="prefix">A string value that will be used as an argument for the <c>putenv</c>
      function.  If this is <see langword="null"/>, the <c>putenv</c> function is not invoked.  Usually used
      to set the value of the <c>TESSDATA_PREFIX</c> environment variable.</param>
01453 /// <param name="language">The name of the language model file to use for the OCR.</param>
01454 /// <param name="callback">A progress callback function.  This function will be called with an integer
      parameter ranging from 0 to 100 to indicate OCR progress, and should return 0 to continue or 1 to
      abort the OCR process.</param>
01455 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
01456         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01457         internal static extern int GetStructuredTextPageWithOCR(IntPtr ctx, IntPtr list, ref IntPtr
      out_page, ref int out_stext_block_count, float zoom, float x0, float y0, float x1, float y1, string
      prefix, string language, [MarshalAs(UnmanagedType.FunctionPtr)] ProgressCallback callback);
01458
01459 /// <summary>
01460 /// Free a native structured text page and its associated resources.
01461 /// </summary>
01462 /// <param name="ctx"></param>
01463 /// <param name="page"></param>
01464 /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
      occurred.</returns>
01465         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01466         internal static extern int DisposeStructuredTextPage(IntPtr ctx, IntPtr page);
01467
01468 /// <summary>
01469 /// Redirect the standard output and standard error to named pipes with the specified names.  On
      Windows, these are actually named pipes; on Linux and macOS, these are Unix sockets (matching the
      behaviour of System.IO.Pipes).  Note that this has side-effects.
01470 /// </summary>
01471 /// <param name="stdoutFD">When the method returns, this variable will contain the file descriptor
      corresponding to the "real" stdout.</param>
01472 /// <param name="stderrFD">When the method returns, this variable will contain the file descriptor
      corresponding to the "real" stderr.</param>
01473 /// <param name="stdoutPipeName">The name of the pipe where stdout will be redirected.  On Windows,
      this should be of the form "\\.\pipe\xxx", while on Linux and macOS it should be an absolute file path
      (maximum length 107/108 characters).</param>
01474 /// <param name="stderrPipeName">The name of the pipe where stderr will be redirected.  On Windows,
      this should be of the form "\\.\pipe\xxx", while on Linux and macOS it should be an absolute file path
      (maximum length 107/108 characters).</param>
01475         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01476         internal static extern void RedirectOutput(out int stdoutFD, out int stderrFD, string
      stdoutPipeName, string stderrPipeName);
01477
01478 /// <summary>
01479 /// Write the specified <paramref name="text"/> to a file descriptor.  Use 1 for stdout and 2 for
      stderr (which may have been redirected).
01480 /// </summary>
01481 /// <param name="fileDescriptor">The file descriptor on which to write.</param>
01482 /// <param name="text">The text to write.</param>
01483 /// <param name="length">The length of the text to write (i.e., text.Length).</param>
01484         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01485         internal static extern void WriteToFileDescriptor(int fileDescriptor, string text, int
      length);
01486
01487 /// <summary>
01488 /// Reset the standard output and standard error (or redirect them to the specified file descriptors,
      theoretically).  Use with the <paramref name="stdoutFD"/> and <paramref name="stderrFD"/> returned by
      <see cref="RedirectOutput"/> to undo what it did.
01489 /// </summary>
01490 /// <param name="stdoutFD">The file descriptor corresponding to the "real" stdout.</param>
01491 /// <param name="stderrFD">The file descriptor corresponding to the "real" stderr.</param>
01492         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01493         internal static extern void ResetOutput(int stdoutFD, int stderrFD);
01494
01495 /// <summary>
```

```
01496 /// Unlocks a document with a password.
01497 /// </summary>
01498 /// <param name="ctx">A context to hold the exception stack and the cached resources.</param>
01499 /// <param name="doc">The document that needs to be unlocked.</param>
01500 /// <param name="password">The password to unlock the document.</param>
01501 /// <returns>0 if the document could not be unlocked, 1 if the document did not require unlocking in
      the first place, 2 if the document was unlocked using the user password and 4 if the document was
      unlocked using the owner password.</returns>
01502         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01503         internal static extern int UnlockWithPassword(IntPtr ctx, IntPtr doc, string password);
01504
01505 /// <summary>
01506 /// Checks whether a password is required to open the document.
01507 /// </summary>
01508 /// <param name="ctx">A context to hold the exception stack and the cached resources.</param>
01509 /// <param name="doc">The document that needs to be checked.</param>
01510 /// <returns>0 if a password is not needed, 1 if a password is needed.</returns>
01511         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01512         internal static extern int CheckIfPasswordNeeded(IntPtr ctx, IntPtr doc);
01513
01514 /// <summary>
01515 /// Returns the current permissions for the document.  Note that these are not actually enforced.
01516 /// </summary>
01517 /// <param name="ctx">A context to hold the exception stack and the cached resources.</param>
01518 /// <param name="doc">The document whose permissions need to be checked.</param>
01519 /// <returns>An integer with bit 0 set if the document can be printed, bit 1 set if it can be copied,
      bit 2 set if it can be edited, and bit 3 set if it can be annotated.</returns>
01520         [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01521         internal static extern int GetPermissions(IntPtr ctx, IntPtr doc);
01522     }
01523 }
```

8.5 MuPDFContext.cs

```
00001 /*
00002 MuPDFCore - A set of multiplatform .NET Core bindings for MuPDF.
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 System;
00019
00020 namespace MuPDFCore
00021 {
00022 /// <summary>
00023 /// A wrapper around a MuPDF context object, which contains the exception stack and the resource cache
      store.
00024 /// </summary>
00025     public class MuPDFContext :  IDisposable
00026     {
00027 /// <summary>
00028 /// A pointer to the native context object.
00029 /// </summary>
00030         internal readonly IntPtr NativeContext;
00031
00032 /// <summary>
00033 /// The current size in bytes of the resource cache store.  Read-only.
00034 /// </summary>
00035         public long StoreSize
00036         {
00037             get
00038             {
00039                 return (long)NativeMethods.GetCurrentStoreSize(this.NativeContext);
00040             }
00041         }
00042
00043 /// <summary>
00044 /// The maximum size in bytes of the resource cache store.  Read-only.
00045 /// </summary>
00046         public long StoreMaxSize
00047         {
00048             get
```

```
00049             {
00050                 return (long)NativeMethods.GetMaxStoreSize(this.NativeContext);
00051             }
00052         }
00053 
00054 /// <summary>
00055 /// Sets the current anti-aliasing level.  Changing this value will affect both
00056 /// the <see cref="TextAntiAliasing"/> and the <see cref="GraphicsAntiAliasing"/>.
00057 /// </summary>
00058         public int AntiAliasing
00059         {
00060             set
00061             {
00062                 if (value < 0 || value > 8)
00063                 {
00064                     throw new ArgumentOutOfRangeException(nameof(value), value, "The anti-aliasing
      level must range between 0 and 8 (inclusive).");
00065                 }
00066 
00067                 NativeMethods.SetAALevel(this.NativeContext, value, -1, -1);
00068             }
00069         }
00070 
00071 /// <summary>
00072 /// Gets or sets the current text anti-aliasing level.
00073 /// </summary>
00074         public int TextAntiAliasing
00075         {
00076             get
00077             {
00078                 NativeMethods.GetAALevel(this.NativeContext, out _, out _, out int tbr);
00079                 return tbr;
00080             }
00081 
00082             set
00083             {
00084                 if (value < 0 || value > 8)
00085                 {
00086                     throw new ArgumentOutOfRangeException(nameof(value), value, "The anti-aliasing
      level must range between 0 and 8 (inclusive).");
00087                 }
00088 
00089                 NativeMethods.SetAALevel(this.NativeContext, -1, -1, value);
00090             }
00091         }
00092 
00093 /// <summary>
00094 /// Gets or sets the current graphics anti-aliasing level.
00095 /// </summary>
00096         public int GraphicsAntiAliasing
00097         {
00098             get
00099             {
00100                 NativeMethods.GetAALevel(this.NativeContext, out _, out int tbr, out _);
00101                 return tbr;
00102             }
00103 
00104             set
00105             {
00106                 if (value < 0 || value > 8)
00107                 {
00108                     throw new ArgumentOutOfRangeException(nameof(value), value, "The anti-aliasing
      level must range between 0 and 8 (inclusive).");
00109                 }
00110 
00111                 int prevTxt = this.TextAntiAliasing;
00112                 NativeMethods.SetAALevel(this.NativeContext, -1, value, prevTxt);
00113             }
00114         }
00115 
00116 /// <summary>
00117 /// Create a new <see cref="MuPDFContext"/> instance with the specified cache store size.
00118 /// </summary>
00119 /// <param name="storeSize">The maximum size in bytes of the resource cache store.  The default value
      is 256 MiB.</param>
00120         public MuPDFContext(uint storeSize = 256 << 20)
00121         {
00122             ExitCodes result = (ExitCodes)NativeMethods.CreateContext((ulong)storeSize, ref
      NativeContext);
00123 
00124             switch (result)
00125             {
00126                 case ExitCodes.EXIT_SUCCESS:
00127                     break;
00128                 case ExitCodes.ERR_CANNOT_CREATE_CONTEXT:
00129                     throw new MuPDFException("Cannot create MuPDF context", result);
00130                 case ExitCodes.ERR_CANNOT_REGISTER_HANDLERS:
```

```
00131                     throw new MuPDFException("Cannot register document handlers", result);
00132                 default:
00133                     throw new MuPDFException("Unknown error", result);
00134             }
00135         }
00136
00137 /// <summary>
00138 /// Wrap an existing pointer to a native MuPDF context object.
00139 /// </summary>
00140 /// <param name="nativeContext">The pointer to the native context that should be used.</param>
00141         internal MuPDFContext(IntPtr nativeContext)
00142         {
00143             this.NativeContext = nativeContext;
00144         }
00145
00146 /// <summary>
00147 /// Evict all items from the resource cache store (freeing the memory where they were held).
00148 /// </summary>
00149         public void ClearStore()
00150         {
00151             NativeMethods.EmptyStore(this.NativeContext);
00152         }
00153
00154 /// <summary>
00155 /// Evict items from the resource cache store (freeing the memory where they were held) until the the
      size of the store drops to the specified fraction of the current size.
00156 /// </summary>
00157 /// <param name="fraction">The fraction of the current size that constitutes the target size of the
      store.  If this is &lt;= 0, the cache is cleared.  If this is &gt;= 1, nothing happens.</param>
00158         public void ShrinkStore(double fraction)
00159         {
00160             if (fraction <= 0)
00161             {
00162                 ClearStore();
00163             }
00164             else if (Math.Round(fraction * 100) < 100)
00165             {
00166                 NativeMethods.ShrinkStore(this.NativeContext, (uint)Math.Round(fraction * 100));
00167             }
00168         }
00169
00170         private bool disposedValue;
00171
00172 ///<inheritdoc/>
00173         protected virtual void Dispose(bool disposing)
00174         {
00175             if (!disposedValue)
00176             {
00177                 if (disposing)
00178                 {
00179                     // TODO: eliminare lo stato gestito (oggetti gestiti)
00180                 }
00181
00182                 NativeMethods.DisposeContext(NativeContext);
00183                 disposedValue = true;
00184             }
00185         }
00186
00187 ///<inheritdoc/>
00188         ~MuPDFContext()
00189         {
00190             Dispose(disposing:  false);
00191         }
00192
00193 ///<inheritdoc/>
00194         public void Dispose()
00195         {
00196             Dispose(disposing:  true);
00197             GC.SuppressFinalize(this);
00198         }
00199     }
00200 }
```

8.6 MuPDFDisplayList.cs

```
00001 /*
00002 MuPDFCore - A set of multiplatform .NET Core bindings for MuPDF.
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 System;
00019
00020 namespace MuPDFCore
00021 {
00022 /// <summary>
00023 /// A wrapper around a MuPDF display list object, which contains the necessary informations to render
      a page to an image.
00024 /// </summary>
00025     internal class MuPDFDisplayList :  IDisposable
00026     {
00027 /// <summary>
00028 /// The context that owns the document that was used to create this display list.
00029 /// </summary>
00030         private readonly MuPDFContext OwnerContext;
00031
00032 /// <summary>
00033 /// A pointer to the native display list object.
00034 /// </summary>
00035         readonly internal IntPtr NativeDisplayList;
00036
00037 /// <summary>
00038 /// The display list's bounds in page units.  Read-only.
00039 /// </summary>
00040         public Rectangle Bounds { get; }
00041
00042 /// <summary>
00043 /// Create a new <see cref="MuPDFDisplayList"/> instance from the specified page.
00044 /// </summary>
00045 /// <param name="context">The context that owns the document from which the page was taken.</param>
00046 /// <param name="page">The page from which the display list should be generated.</param>
00047 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included in the display list that is generated.  Otherwise, only the page contents are
      included.</param>
00048         public MuPDFDisplayList(MuPDFContext context, MuPDFPage page, bool includeAnnotations = true)
00049         {
00050             this.OwnerContext = context;
00051
00052             float x0 = 0;
00053             float y0 = 0;
00054             float x1 = 0;
00055             float y1 = 0;
00056
00057             ExitCodes result = (ExitCodes)NativeMethods.GetDisplayList(context.NativeContext,
      page.NativePage, includeAnnotations ?  1 :  0, ref NativeDisplayList, ref x0, ref y0, ref x1, ref y1);
00058
00059             switch (result)
00060             {
00061                 case ExitCodes.EXIT_SUCCESS:
00062                     break;
00063                 case ExitCodes.ERR_CANNOT_RENDER:
00064                     throw new MuPDFException("Cannot render page", result);
00065                 default:
00066                     throw new MuPDFException("Unknown error", result);
00067             }
00068
00069             this.Bounds = new Rectangle(Math.Round(x0 * page.OwnerDocument.ImageXRes / 72.0 * 1000) /
      1000, Math.Round(y0 * page.OwnerDocument.ImageYRes / 72.0 * 1000) / 1000, Math.Round(x1 *
      page.OwnerDocument.ImageXRes / 72.0 * 1000) / 1000, Math.Round(y1 * page.OwnerDocument.ImageYRes /
      72.0 * 1000) / 1000);
00070         }
00071
00072         private bool disposedValue;
00073
00074         protected virtual void Dispose(bool disposing)
00075         {
00076             if (!disposedValue)
00077             {
00078                 NativeMethods.DisposeDisplayList(OwnerContext.NativeContext, NativeDisplayList);
00079                 disposedValue = true;
00080             }
00081         }
00082
00083         ~MuPDFDisplayList()
00084         {
00085             Dispose(disposing:  false);
00086         }
00087
00088         public void Dispose()
```

```
00089         {
00090             Dispose(disposing:  true);
00091             GC.SuppressFinalize(this);
00092         }
00093     }
00094 }
```

8.7 MuPDFDocument.cs

```
00001 /*
00002 MuPDFCore - A set of multiplatform .NET Core bindings for MuPDF.
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 System;
00019 using System.IO;
00020 using System.Runtime.InteropServices;
00021 using System.Text;
00022 using System.Threading;
00023 using System.Threading.Tasks;
00024
00025 namespace MuPDFCore
00026 {
00027 /// <summary>
00028 /// A wrapper over a MuPDF document object, which contains possibly multiple pages.
00029 /// </summary>
00030     public class MuPDFDocument :  IDisposable
00031     {
00032 /// <summary>
00033 /// If the document is an image, the horizontal resolution of the image.  Otherwise, 72.
00034 /// </summary>
00035         internal double ImageXRes = double.NaN;
00036
00037 /// <summary>
00038 /// If the document is an image, the vertical resolution of the image.  Otherwise, 72.
00039 /// </summary>
00040         internal double ImageYRes = double.NaN;
00041
00042 /// <summary>
00043 /// File extensions corresponding to the supported input formats.
00044 /// </summary>
00045         private static readonly string[] FileTypeMagics = new[]
00046         {
00047             ".pdf",
00048             ".xps",
00049             ".cbz",
00050             ".png",
00051             ".jpg",
00052             ".bmp",
00053             ".gif",
00054             ".tif",
00055             ".pnm",
00056             ".pam",
00057             ".epub",
00058             ".fb2",
00059             ".mobi",
00060             ".html"
00061         };
00062
00063 /// <summary>
00064 /// An <see cref="IDisposable"/> with a value of null.
00065 /// </summary>
00066         private static IDisposable NullDataHolder = null;
00067
00068 /// <summary>
00069 /// The context that owns this document.
00070 /// </summary>
00071         private readonly MuPDFContext OwnerContext;
00072
00073 /// <summary>
00074 /// A pointer to the native document object.
```

```
00075 /// </summary>
00076         internal readonly IntPtr NativeDocument;
00077 
00078 /// <summary>
00079 /// A pointer to the native stream that was used to create this document (if any).
00080 /// </summary>
00081         private readonly IntPtr NativeStream = IntPtr.Zero;
00082 
00083 /// <summary>
00084 /// The number of pages in the document.
00085 /// </summary>
00086         private int PageCount;
00087 
00088 /// <summary>
00089 /// An <see cref="IDisposable"/> that will be disposed together with this object.
00090 /// </summary>
00091         private readonly IDisposable DataHolder = null;
00092 
00093 /// <summary>
00094 /// A <see cref="GCHandle"/> that will be freed when this object is disposed.
00095 /// </summary>
00096         private GCHandle?  DataHandle = null;
00097 
00098 /// <summary>
00099 /// An array of <see cref="MuPDFDisplayList"/>, one for each page in the document.
00100 /// </summary>
00101         private MuPDFDisplayList[] DisplayLists;
00102 
00103 /// <summary>
00104 /// The pages contained in the document.
00105 /// </summary>
00106         public MuPDFPageCollection Pages { get; private set; }
00107 
00108 /// <summary>
00109 /// Defines whether the images resulting from rendering operations should be clipped to the page
      boundaries.
00110 /// </summary>
00111         public bool ClipToPageBounds { get; set; } = true;
00112 
00113 /// <summary>
00114 /// Describes the encryption state of the document.
00115 /// </summary>
00116         public EncryptionState EncryptionState { get; private set; }
00117 
00118 /// <summary>
00119 /// Describes the restriction state of the document.
00120 /// </summary>
00121         public RestrictionState RestrictionState { get; private set; }
00122 
00123 /// <summary>
00124 /// Describes the operations that are restricted on the document.  This is not actually enforced by
      the library,
00125 /// but library users should only allow these operations if the document has been unlocked with the
      owner password
00126 /// (i.e.  if <see cref="RestrictionState"/> is <see cref="RestrictionState.Unlocked"/>).
00127 /// </summary>
00128         public DocumentRestrictions Restrictions { get; private set; }
00129 
00130 /// <summary>
00131 /// Create a new <see cref="MuPDFDocument"/> from data bytes accessible through the specified pointer.
00132 /// </summary>
00133 /// <param name="context">The context that will own this document.</param>
00134 /// <param name="dataAddress">A pointer to the data bytes that make up the document.</param>
00135 /// <param name="dataLength">The number of bytes to read from the specified address.</param>
00136 /// <param name="fileType">The type of the document to read.</param>
00137         public MuPDFDocument(MuPDFContext context, IntPtr dataAddress, long dataLength, InputFileTypes
      fileType) :  this(context, dataAddress, dataLength, fileType, ref NullDataHolder) { }
00138 
00139 /// <summary>
00140 /// Create a new <see cref="MuPDFDocument"/> from data bytes accessible through the specified pointer.
00141 /// </summary>
00142 /// <param name="context">The context that will own this document.</param>
00143 /// <param name="dataAddress">A pointer to the data bytes that make up the document.</param>
00144 /// <param name="dataLength">The number of bytes to read from the specified address.</param>
00145 /// <param name="fileType">The type of the document to read.</param>
00146 /// <param name="dataHolder">An <see cref="IDisposable"/> that will be disposed when the <see
      cref="MuPDFDocument"/> is disposed.</param>
00147         public MuPDFDocument(MuPDFContext context, IntPtr dataAddress, long dataLength, InputFileTypes
      fileType, ref IDisposable dataHolder)
00148         {
00149             bool isImage = fileType == InputFileTypes.BMP || fileType == InputFileTypes.GIF ||
      fileType == InputFileTypes.JPEG || fileType == InputFileTypes.PAM || fileType == InputFileTypes.PNG ||
      fileType == InputFileTypes.PNM || fileType == InputFileTypes.TIFF;
00150 
00151             this.OwnerContext = context;
00152 
00153             float xRes = 0;
```

```
00154             float yRes = 0;
00155
00156             ExitCodes result =
      (ExitCodes)NativeMethods.CreateDocumentFromStream(context.NativeContext, dataAddress,
      (ulong)dataLength, FileTypeMagics[(int)fileType], isImage ?  1 :  0, ref NativeDocument, ref
      NativeStream, ref PageCount, ref xRes, ref yRes);
00157
00158             if (xRes > 72)
00159             {
00160                 this.ImageXRes = xRes;
00161             }
00162             else
00163             {
00164                 this.ImageXRes = 72;
00165             }
00166
00167             if (yRes > 72)
00168             {
00169                 this.ImageYRes = yRes;
00170             }
00171             else
00172             {
00173                 this.ImageYRes = 72;
00174             }
00175
00176             this.DataHolder = dataHolder;
00177
00178             switch (result)
00179             {
00180                 case ExitCodes.EXIT_SUCCESS:
00181                     break;
00182                 case ExitCodes.ERR_CANNOT_OPEN_STREAM:
00183                     throw new MuPDFException("Cannot open data stream", result);
00184                 case ExitCodes.ERR_CANNOT_OPEN_FILE:
00185                     throw new MuPDFException("Cannot open document", result);
00186                 case ExitCodes.ERR_CANNOT_COUNT_PAGES:
00187                     throw new MuPDFException("Cannot count pages", result);
00188                 default:
00189                     throw new MuPDFException("Unknown error", result);
00190             }
00191
00192             if (NativeMethods.CheckIfPasswordNeeded(context.NativeContext, this.NativeDocument) != 0)
00193             {
00194                 this.EncryptionState = EncryptionState.Encrypted;
00195             }
00196             else
00197             {
00198                 this.EncryptionState = EncryptionState.Unencrypted;
00199             }
00200
00201             int permissions = NativeMethods.GetPermissions(context.NativeContext,
      this.NativeDocument);
00202
00203             int restrictions = 0;
00204
00205             if ((permissions & 1) == 0)
00206             {
00207                 restrictions |= 1;
00208             }
00209
00210             if ((permissions & 2) == 0)
00211             {
00212                 restrictions |= 2;
00213             }
00214
00215             if ((permissions & 4) == 0)
00216             {
00217                 restrictions |= 4;
00218             }
00219
00220             if ((permissions & 8) == 0)
00221             {
00222                 restrictions |= 8;
00223             }
00224
00225             if (restrictions == 0)
00226             {
00227                 this.Restrictions = DocumentRestrictions.None;
00228                 this.RestrictionState = RestrictionState.Unrestricted;
00229             }
00230             else
00231             {
00232                 this.Restrictions = (DocumentRestrictions)restrictions;
00233                 this.RestrictionState = RestrictionState.Restricted;
00234             }
00235
00236             Pages = new MuPDFPageCollection(context, this, PageCount);
```

```
00237             DisplayLists = new MuPDFDisplayList[PageCount];
00238         }
00239
00240 /// <summary>
00241 /// Create a new <see cref="MuPDFDocument"/> from an array of bytes.
00242 /// </summary>
00243 /// <param name="context">The context that will own this document.</param>
00244 /// <param name="data">An array containing the data bytes that make up the document.  This must not be
      altered until after the <see cref="MuPDFDocument"/> has been disposed!
00245 /// The address of the array will be pinned, which may cause degradation in the Garbage Collector's
      performance, and is thus only advised for short-lived documents.  To avoid this issue, marshal the
      bytes to unmanaged memory and use one of the <see cref="IntPtr"/> constructors.</param>
00246 /// <param name="fileType">The type of the document to read.</param>
00247         public MuPDFDocument(MuPDFContext context, byte[] data, InputFileTypes fileType)
00248         {
00249             bool isImage = fileType == InputFileTypes.BMP || fileType == InputFileTypes.GIF ||
      fileType == InputFileTypes.JPEG || fileType == InputFileTypes.PAM || fileType == InputFileTypes.PNG ||
      fileType == InputFileTypes.PNM || fileType == InputFileTypes.TIFF;
00250
00251             this.OwnerContext = context;
00252
00253             DataHandle = GCHandle.Alloc(data, GCHandleType.Pinned);
00254             IntPtr dataAddress = DataHandle.Value.AddrOfPinnedObject();
00255             ulong dataLength = (ulong)data.Length;
00256
00257             float xRes = 0;
00258             float yRes = 0;
00259
00260             ExitCodes result =
      (ExitCodes)NativeMethods.CreateDocumentFromStream(context.NativeContext, dataAddress, dataLength,
      FileTypeMagics[(int)fileType], isImage ?  1 :  0, ref NativeDocument, ref NativeStream, ref PageCount,
      ref xRes, ref yRes);
00261
00262             if (xRes > 72)
00263             {
00264                 this.ImageXRes = xRes;
00265             }
00266             else
00267             {
00268                 this.ImageXRes = 72;
00269             }
00270
00271             if (yRes > 72)
00272             {
00273                 this.ImageYRes = yRes;
00274             }
00275             else
00276             {
00277                 this.ImageYRes = 72;
00278             }
00279
00280             switch (result)
00281             {
00282                 case ExitCodes.EXIT_SUCCESS:
00283                     break;
00284                 case ExitCodes.ERR_CANNOT_OPEN_STREAM:
00285                     throw new MuPDFException("Cannot open data stream", result);
00286                 case ExitCodes.ERR_CANNOT_OPEN_FILE:
00287                     throw new MuPDFException("Cannot open document", result);
00288                 case ExitCodes.ERR_CANNOT_COUNT_PAGES:
00289                     throw new MuPDFException("Cannot count pages", result);
00290                 default:
00291                     throw new MuPDFException("Unknown error", result);
00292             }
00293
00294             if (NativeMethods.CheckIfPasswordNeeded(context.NativeContext, this.NativeDocument) != 0)
00295             {
00296                 this.EncryptionState = EncryptionState.Encrypted;
00297             }
00298             else
00299             {
00300                 this.EncryptionState = EncryptionState.Unencrypted;
00301             }
00302
00303             int permissions = NativeMethods.GetPermissions(context.NativeContext,
      this.NativeDocument);
00304
00305             int restrictions = 0;
00306
00307             if ((permissions & 1) == 0)
00308             {
00309                 restrictions |= 1;
00310             }
00311
00312             if ((permissions & 2) == 0)
00313             {
00314                 restrictions |= 2;
```

```
00315             }
00316
00317             if ((permissions & 4) == 0)
00318             {
00319                 restrictions |= 4;
00320             }
00321
00322             if ((permissions & 8) == 0)
00323             {
00324                 restrictions |= 8;
00325             }
00326
00327             if (restrictions == 0)
00328             {
00329                 this.Restrictions = DocumentRestrictions.None;
00330                 this.RestrictionState = RestrictionState.Unrestricted;
00331             }
00332             else
00333             {
00334                 this.Restrictions = (DocumentRestrictions)restrictions;
00335                 this.RestrictionState = RestrictionState.Restricted;
00336             }
00337
00338             Pages = new MuPDFPageCollection(context, this, PageCount);
00339             DisplayLists = new MuPDFDisplayList[PageCount];
00340         }
00341
00342 /// <summary>
00343 /// Create a new <see cref="MuPDFDocument"/> from a <see cref="MemoryStream"/>.
00344 /// </summary>
00345 /// <param name="context">The context that will own this document.</param>
00346 /// <param name="data">The <see cref="MemoryStream"/> containing the data that makes up the document.
      This will be disposed when the <see cref="MuPDFDocument"/> has been disposed and must not be disposed
      externally!
00347 /// The address of the <see cref="MemoryStream"/>'s buffer will be pinned, which may cause degradation
      in the Garbage Collector's performance, and is thus only advised for short-lived documents.  To avoid
      this issue, marshal the bytes to unmanaged memory and use one of the <see cref="IntPtr"/>
      constructors.</param>
00348 /// <param name="fileType">The type of the document to read.</param>
00349         public MuPDFDocument(MuPDFContext context, ref MemoryStream data, InputFileTypes fileType)
00350         {
00351             bool isImage = fileType == InputFileTypes.BMP || fileType == InputFileTypes.GIF ||
      fileType == InputFileTypes.JPEG || fileType == InputFileTypes.PAM || fileType == InputFileTypes.PNG ||
      fileType == InputFileTypes.PNM || fileType == InputFileTypes.TIFF;
00352
00353             this.OwnerContext = context;
00354
00355             int origin = (int)data.Seek(0, SeekOrigin.Begin);
00356             ulong dataLength = (ulong)data.Length;
00357             byte[] dataBytes = data.GetBuffer();
00358
00359             DataHandle = GCHandle.Alloc(dataBytes, GCHandleType.Pinned);
00360             IntPtr dataAddress = IntPtr.Add(DataHandle.Value.AddrOfPinnedObject(), origin);
00361
00362             DataHolder = data;
00363
00364             float xRes = 0;
00365             float yRes = 0;
00366
00367             ExitCodes result =
      (ExitCodes)NativeMethods.CreateDocumentFromStream(context.NativeContext, dataAddress, dataLength,
      FileTypeMagics[(int)fileType], isImage ?  1 :  0, ref NativeDocument, ref NativeStream, ref PageCount,
      ref xRes, ref yRes);
00368
00369             if (xRes > 72)
00370             {
00371                 this.ImageXRes = xRes;
00372             }
00373             else
00374             {
00375                 this.ImageXRes = 72;
00376             }
00377
00378             if (yRes > 72)
00379             {
00380                 this.ImageYRes = yRes;
00381             }
00382             else
00383             {
00384                 this.ImageYRes = 72;
00385             }
00386
00387
00388             switch (result)
00389             {
00390                 case ExitCodes.EXIT_SUCCESS:
00391                     break;
```

```
00392                 case ExitCodes.ERR_CANNOT_OPEN_STREAM:
00393                     throw new MuPDFException("Cannot open data stream", result);
00394                 case ExitCodes.ERR_CANNOT_OPEN_FILE:
00395                     throw new MuPDFException("Cannot open document", result);
00396                 case ExitCodes.ERR_CANNOT_COUNT_PAGES:
00397                     throw new MuPDFException("Cannot count pages", result);
00398                 default:
00399                     throw new MuPDFException("Unknown error", result);
00400             }
00401 
00402             if (NativeMethods.CheckIfPasswordNeeded(context.NativeContext, this.NativeDocument) != 0)
00403             {
00404                 this.EncryptionState = EncryptionState.Encrypted;
00405             }
00406             else
00407             {
00408                 this.EncryptionState = EncryptionState.Unencrypted;
00409             }
00410 
00411             int permissions = NativeMethods.GetPermissions(context.NativeContext,
      this.NativeDocument);
00412 
00413             int restrictions = 0;
00414 
00415             if ((permissions & 1) == 0)
00416             {
00417                 restrictions |= 1;
00418             }
00419 
00420             if ((permissions & 2) == 0)
00421             {
00422                 restrictions |= 2;
00423             }
00424 
00425             if ((permissions & 4) == 0)
00426             {
00427                 restrictions |= 4;
00428             }
00429 
00430             if ((permissions & 8) == 0)
00431             {
00432                 restrictions |= 8;
00433             }
00434 
00435             if (restrictions == 0)
00436             {
00437                 this.Restrictions = DocumentRestrictions.None;
00438                 this.RestrictionState = RestrictionState.Unrestricted;
00439             }
00440             else
00441             {
00442                 this.Restrictions = (DocumentRestrictions)restrictions;
00443                 this.RestrictionState = RestrictionState.Restricted;
00444             }
00445 
00446             Pages = new MuPDFPageCollection(context, this, PageCount);
00447             DisplayLists = new MuPDFDisplayList[PageCount];
00448         }
00449 
00450 /// <summary>
00451 /// Create a new <see cref="MuPDFDocument"/> from a file.
00452 /// </summary>
00453 /// <param name="context">The context that will own this document.</param>
00454 /// <param name="fileName">The path to the file to open.</param>
00455         public MuPDFDocument(MuPDFContext context, string fileName)
00456         {
00457             bool isImage;
00458 
00459             string extension = Path.GetExtension(fileName).ToLowerInvariant();
00460 
00461             switch (extension)
00462             {
00463                 case ".bmp":
00464                 case ".dib":
00465 
00466                 case ".gif":
00467 
00468                 case ".jpg":
00469                 case ".jpeg":
00470                 case ".jpe":
00471                 case ".jif":
00472                 case ".jfif":
00473                 case ".jfi":
00474 
00475                 case ".pam":
00476                 case ".pbm":
00477                 case ".pgm":
```

```
00478                 case ".ppm":
00479                 case ".pnm":
00480
00481                 case ".png":
00482
00483                 case ".tif":
00484                 case ".tiff":
00485                     isImage = true;
00486                     break;
00487                 default:
00488                     isImage = false;
00489                     break;
00490             }
00491
00492
00493             this.OwnerContext = context;
00494
00495             float xRes = 0;
00496             float yRes = 0;
00497
00498             ExitCodes result;
00499
00500             using (UTF8EncodedString encodedFileName = new UTF8EncodedString(fileName))
00501             {
00502                 result = (ExitCodes)NativeMethods.CreateDocumentFromFile(context.NativeContext,
      encodedFileName.Address, isImage ?  1 :  0, ref NativeDocument, ref PageCount, ref xRes, ref yRes);
00503             }
00504
00505             if (xRes > 72)
00506             {
00507                 this.ImageXRes = xRes;
00508             }
00509             else
00510             {
00511                 this.ImageXRes = 72;
00512             }
00513
00514             if (yRes > 72)
00515             {
00516                 this.ImageYRes = yRes;
00517             }
00518             else
00519             {
00520                 this.ImageYRes = 72;
00521             }
00522
00523             switch (result)
00524             {
00525                 case ExitCodes.EXIT_SUCCESS:
00526                     break;
00527                 case ExitCodes.ERR_CANNOT_OPEN_FILE:
00528                     throw new MuPDFException("Cannot open document", result);
00529                 case ExitCodes.ERR_CANNOT_COUNT_PAGES:
00530                     throw new MuPDFException("Cannot count pages", result);
00531                 default:
00532                     throw new MuPDFException("Unknown error", result);
00533             }
00534
00535             if (NativeMethods.CheckIfPasswordNeeded(context.NativeContext, this.NativeDocument) != 0)
00536             {
00537                 this.EncryptionState = EncryptionState.Encrypted;
00538             }
00539             else
00540             {
00541                 this.EncryptionState = EncryptionState.Unencrypted;
00542             }
00543
00544             int permissions = NativeMethods.GetPermissions(context.NativeContext,
      this.NativeDocument);
00545
00546             int restrictions = 0;
00547
00548             if ((permissions & 1) == 0)
00549             {
00550                 restrictions |= 1;
00551             }
00552
00553             if ((permissions & 2) == 0)
00554             {
00555                 restrictions |= 2;
00556             }
00557
00558             if ((permissions & 4) == 0)
00559             {
00560                 restrictions |= 4;
00561             }
00562
```

```
00563             if ((permissions & 8) == 0)
00564             {
00565                 restrictions |= 8;
00566             }
00567 
00568             if (restrictions == 0)
00569             {
00570                 this.Restrictions = DocumentRestrictions.None;
00571                 this.RestrictionState = RestrictionState.Unrestricted;
00572             }
00573             else
00574             {
00575                 this.Restrictions = (DocumentRestrictions)restrictions;
00576                 this.RestrictionState = RestrictionState.Restricted;
00577             }
00578 
00579             Pages = new MuPDFPageCollection(context, this, PageCount);
00580             DisplayLists = new MuPDFDisplayList[PageCount];
00581         }
00582 
00583 /// <summary>
00584 /// Discard all the display lists that have been loaded from the document, possibly freeing some
      memory in the case of a huge document.
00585 /// </summary>
00586         public void ClearCache()
00587         {
00588             for (int i = 0; i < PageCount; i++)
00589             {
00590                 DisplayLists[i]?.Dispose();
00591                 DisplayLists[i] = null;
00592             }
00593         }
00594 
00595 /// <summary>
00596 /// Sets the document layout for reflowable document types (e.g., HTML, MOBI). Does not have any
      effect for documents with a fixed layout (e.g., PDF).
00597 /// </summary>
00598 /// <param name="width">The width of each page, in points.  Must be &gt; 0.</param>
00599 /// <param name="height">The height of each page, in points.  Must be &gt; 0.</param>
00600 /// <param name="em">The default font size, in points.</param>
00601         public void Layout(float width, float height, float em)
00602         {
00603             if (width <= 0)
00604             {
00605                 throw new ArgumentOutOfRangeException(nameof(width), width, "The page width must be
      greater than 0!");
00606             }
00607 
00608             if (height <= 0)
00609             {
00610                 throw new ArgumentOutOfRangeException(nameof(height), height, "The page height must be
      greater than 0!");
00611             }
00612 
00613             this.ClearCache();
00614             this.Pages.Dispose();
00615 
00616             NativeMethods.LayoutDocument(this.OwnerContext.NativeContext, this.NativeDocument, width,
      height, em, out int pageCount);
00617 
00618             this.PageCount = pageCount;
00619             this.Pages = new MuPDFPageCollection(this.OwnerContext, this, PageCount);
00620             this.DisplayLists = new MuPDFDisplayList[PageCount];
00621         }
00622 
00623 /// <summary>
00624 /// Sets the document layout for reflowable document types (e.g., HTML, MOBI), so that the document is
      rendered to a single
00625 /// page, as tall as necessary.  Does not have any effect for documents with a fixed layout (e.g.,
      PDF).
00626 /// </summary>
00627 /// <param name="width">The width of each page, in points.  Must be &gt; 0.</param>
00628 /// <param name="em">The default font size, in points.</param>
00629         public void LayoutSinglePage(float width, float em)
00630         {
00631             if (width <= 0)
00632             {
00633                 throw new ArgumentOutOfRangeException(nameof(width), width, "The page width must be
      greater than 0!");
00634             }
00635 
00636             this.ClearCache();
00637             this.Pages.Dispose();
00638 
00639             NativeMethods.LayoutDocument(this.OwnerContext.NativeContext, this.NativeDocument, width,
      0, em, out int pageCount);
00640 
```

```
00641             this.PageCount = pageCount;
00642             this.Pages = new MuPDFPageCollection(this.OwnerContext, this, PageCount);
00643             this.DisplayLists = new MuPDFDisplayList[PageCount];
00644         }
00645 
00646 /// <summary>
00647 /// Render (part of) a page to an array of bytes.
00648 /// </summary>
00649 /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
00650 /// <param name="region">The region of the page to render in page units.</param>
00651 /// <param name="zoom">The scale at which the page will be rendered.  This will determine the size in
      pixel of the image.</param>
00652 /// <param name="pixelFormat">The format of the pixel data.</param>
00653 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included in the display list that is generated.  Otherwise, only the page contents are
      included.</param>
00654 /// <returns>A byte array containing the raw values for the pixels of the rendered image.</returns>
00655         public byte[] Render(int pageNumber, Rectangle region, double zoom, PixelFormats pixelFormat,
      bool includeAnnotations = true)
00656         {
00657             if (this.EncryptionState == EncryptionState.Encrypted)
00658             {
00659                 throw new DocumentLockedException("A password is necessary to render the document!");
00660             }
00661 
00662             int bufferSize = MuPDFDocument.GetRenderedSize(region, zoom, pixelFormat);
00663 
00664             byte[] buffer = new byte[bufferSize];
00665 
00666             GCHandle bufferHandle = GCHandle.Alloc(buffer, GCHandleType.Pinned);
00667             IntPtr bufferPointer = bufferHandle.AddrOfPinnedObject();
00668 
00669             try
00670             {
00671                 Render(pageNumber, region, zoom, pixelFormat, bufferPointer, includeAnnotations);
00672             }
00673             finally
00674             {
00675                 bufferHandle.Free();
00676             }
00677 
00678             return buffer;
00679         }
00680 
00681 /// <summary>
00682 /// Render a page to an array of bytes.
00683 /// </summary>
00684 /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
00685 /// <param name="zoom">The scale at which the page will be rendered.  This will determine the size in
      pixel of the image.</param>
00686 /// <param name="pixelFormat">The format of the pixel data.</param>
00687 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included in the display list that is generated.  Otherwise, only the page contents are
      included.</param>
00688 /// <returns>A byte array containing the raw values for the pixels of the rendered image.</returns>
00689         public byte[] Render(int pageNumber, double zoom, PixelFormats pixelFormat, bool
      includeAnnotations = true)
00690         {
00691             if (this.EncryptionState == EncryptionState.Encrypted)
00692             {
00693                 throw new DocumentLockedException("A password is necessary to render the document!");
00694             }
00695 
00696             Rectangle region = this.Pages[pageNumber].Bounds;
00697             return Render(pageNumber, region, zoom, pixelFormat, includeAnnotations);
00698         }
00699 
00700 /// <summary>
00701 /// Render (part of) a page to the specified destination.
00702 /// </summary>
00703 /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
00704 /// <param name="region">The region of the page to render in page units.</param>
00705 /// <param name="zoom">The scale at which the page will be rendered.  This will determine the size in
      pixel of the image.</param>
00706 /// <param name="pixelFormat">The format of the pixel data.</param>
00707 /// <param name="destination">The address of the buffer where the pixel data will be written.  There
      must be enough space available to write the values for all the pixels, otherwise this will fail
      catastrophically!</param>
00708 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included in the display list that is generated.  Otherwise, only the page contents are
      included.</param>
00709         public void Render(int pageNumber, Rectangle region, double zoom, PixelFormats pixelFormat,
      IntPtr destination, bool includeAnnotations = true)
00710         {
00711             if (this.EncryptionState == EncryptionState.Encrypted)
00712             {
00713                 throw new DocumentLockedException("A password is necessary to render the document!");
```

```
00714             }
00715
00716             if (DisplayLists[pageNumber] == null)
00717             {
00718                 DisplayLists[pageNumber] = new MuPDFDisplayList(this.OwnerContext,
      this.Pages[pageNumber], includeAnnotations);
00719             }
00720
00721             if (zoom < 0.000001 | zoom * region.Width <= 0.001 || zoom * region.Height <= 0.001)
00722             {
00723                 throw new ArgumentOutOfRangeException(nameof(zoom), zoom, "The zoom factor is too
      small!");
00724             }
00725
00726             if (this.ImageXRes != 72 || this.ImageYRes != 72)
00727             {
00728                 zoom *= Math.Sqrt(this.ImageXRes * this.ImageYRes) / 72;
00729                 region = new Rectangle(region.X0 * 72 / this.ImageXRes, region.Y0 * 72 /
      this.ImageYRes, region.X1 * 72 / this.ImageXRes, region.Y1 * 72 / this.ImageYRes);
00730             }
00731
00732             float fzoom = (float)zoom;
00733
00734             ExitCodes result =
      (ExitCodes)NativeMethods.RenderSubDisplayList(OwnerContext.NativeContext,
      DisplayLists[pageNumber].NativeDisplayList, region.X0, region.Y0, region.X1, region.Y1, fzoom,
      (int)pixelFormat, destination, IntPtr.Zero);
00735
00736             switch (result)
00737             {
00738                 case ExitCodes.EXIT_SUCCESS:
00739                     break;
00740                 case ExitCodes.ERR_CANNOT_RENDER:
00741                     throw new MuPDFException("Cannot render page", result);
00742                 default:
00743                     throw new MuPDFException("Unknown error", result);
00744             }
00745
00746             RoundedRectangle roundedRegion = region.Round(fzoom);
00747             RoundedSize roundedSize = new RoundedSize(roundedRegion.Width, roundedRegion.Height);
00748
00749             if (pixelFormat == PixelFormats.RGBA || pixelFormat == PixelFormats.BGRA)
00750             {
00751                 Utils.UnpremultiplyAlpha(destination, roundedSize);
00752             }
00753
00754             if (this.ClipToPageBounds &&
      !Pages[pageNumber].Bounds.Contains(DisplayLists[pageNumber].Bounds.Intersect(region)))
00755             {
00756                 Utils.ClipImage(destination, roundedSize, region, Pages[pageNumber].Bounds,
      pixelFormat);
00757             }
00758         }
00759
00760 /// <summary>
00761 /// Render a page to the specified destination.
00762 /// </summary>
00763 /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
00764 /// <param name="zoom">The scale at which the page will be rendered.  This will determine the size in
      pixel of the image.</param>
00765 /// <param name="pixelFormat">The format of the pixel data.</param>
00766 /// <param name="destination">The address of the buffer where the pixel data will be written.  There
      must be enough space available to write the values for all the pixels, otherwise this will fail
      catastrophically!</param>
00767 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included in the display list that is generated.  Otherwise, only the page contents are
      included.</param>
00768         public void Render(int pageNumber, double zoom, PixelFormats pixelFormat, IntPtr destination,
      bool includeAnnotations = true)
00769         {
00770             if (this.EncryptionState == EncryptionState.Encrypted)
00771             {
00772                 throw new DocumentLockedException("A password is necessary to render the document!");
00773             }
00774
00775             Rectangle region = this.Pages[pageNumber].Bounds;
00776             Render(pageNumber, region, zoom, pixelFormat, destination, includeAnnotations);
00777         }
00778
00779 /// <summary>
00780 /// Render (part of) a page to a <see cref="Span{T}">Span</see>&lt;<see cref="byte"/>&gt;.
00781 /// </summary>
00782 /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
00783 /// <param name="region">The region of the page to render in page units.</param>
00784 /// <param name="zoom">The scale at which the page will be rendered.  This will determine the size in
      pixel of the image.</param>
00785 /// <param name="pixelFormat">The format of the pixel data.</param>
```

```
00786 /// <param name="disposable">An <see cref="IDisposable"/> that can be used to free the memory where
      the image is stored.  You should keep track of this and dispose it when you have finished working with
      the image.</param>
00787 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included in the display list that is generated.  Otherwise, only the page contents are
      included.</param>
00788         public Span<byte> Render(int pageNumber, Rectangle region, double zoom, PixelFormats
      pixelFormat, out IDisposable disposable, bool includeAnnotations = true)
00789         {
00790             if (this.EncryptionState == EncryptionState.Encrypted)
00791             {
00792                 throw new DocumentLockedException("A password is necessary to render the document!");
00793             }
00794 
00795             int dataSize = GetRenderedSize(region, zoom, pixelFormat);
00796 
00797             IntPtr destination = Marshal.AllocHGlobal(dataSize);
00798             disposable = new DisposableIntPtr(destination, dataSize);
00799 
00800             this.Render(pageNumber, region, zoom, pixelFormat, destination, includeAnnotations);
00801 
00802             unsafe
00803             {
00804                 return new Span<byte>((void*)destination, dataSize);
00805             }
00806         }
00807 
00808 /// <summary>
00809 /// Render a page to a <see cref="Span{T}">Span</see>&lt;<see cref="byte"/>&gt;.
00810 /// </summary>
00811 /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
00812 /// <param name="zoom">The scale at which the page will be rendered.  This will determine the size in
      pixel of the image.</param>
00813 /// <param name="pixelFormat">The format of the pixel data.</param>
00814 /// <param name="disposable">An <see cref="IDisposable"/> that can be used to free the memory where
      the image is stored.  You should keep track of this and dispose it when you have finished working with
      the image.</param>
00815 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included in the display list that is generated.  Otherwise, only the page contents are
      included.</param>
00816         public Span<byte> Render(int pageNumber, double zoom, PixelFormats pixelFormat, out
      IDisposable disposable, bool includeAnnotations = true)
00817         {
00818             if (this.EncryptionState == EncryptionState.Encrypted)
00819             {
00820                 throw new DocumentLockedException("A password is necessary to render the document!");
00821             }
00822 
00823             Rectangle region = this.Pages[pageNumber].Bounds;
00824             return Render(pageNumber, region, zoom, pixelFormat, out disposable, includeAnnotations);
00825         }
00826 
00827 
00828 /// <summary>
00829 /// Create a new <see cref="MuPDFMultiThreadedPageRenderer"/> that renders the specified page with the
      specified number of threads.
00830 /// </summary>
00831 /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
00832 /// <param name="threadCount">The number of threads to use.  This must be factorisable using only
      powers of 2, 3, 5 or 7.  Otherwise, the biggest number smaller than <paramref name="threadCount"/>
      that satisfies this condition is used.</param>
00833 /// <returns>A <see cref="MuPDFMultiThreadedPageRenderer"/> that can be used to render the specified
      page with the specified number of threads.</returns>
00834 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included in the display list that is generated.  Otherwise, only the page contents are
      included.</param>
00835         public MuPDFMultiThreadedPageRenderer GetMultiThreadedRenderer(int pageNumber, int
      threadCount, bool includeAnnotations = true)
00836         {
00837             if (this.EncryptionState == EncryptionState.Encrypted)
00838             {
00839                 throw new DocumentLockedException("A password is necessary to render the document!");
00840             }
00841 
00842             if (DisplayLists[pageNumber] == null)
00843             {
00844                 DisplayLists[pageNumber] = new MuPDFDisplayList(this.OwnerContext,
      this.Pages[pageNumber], includeAnnotations);
00845             }
00846 
00847             return new MuPDFMultiThreadedPageRenderer(OwnerContext, DisplayLists[pageNumber],
      threadCount, Pages[pageNumber].Bounds, this.ClipToPageBounds, this.ImageXRes, this.ImageYRes);
00848         }
00849 
00850 /// <summary>
00851 /// Determine how many bytes will be necessary to render the specified page at the specified zoom
      level, using the the specified pixel format.
```

```
00852 /// </summary>
00853 /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
00854 /// <param name="zoom">The scale at which the page will be rendered.  This will determine the size in
      pixel of the image.</param>
00855 /// <param name="pixelFormat">The format of the pixels data.</param>
00856 /// <returns>An integer representing the number of bytes that will be necessary to store the pixel
      data of the rendered image.</returns>
00857         public int GetRenderedSize(int pageNumber, double zoom, PixelFormats pixelFormat)
00858         {
00859             if (this.EncryptionState == EncryptionState.Encrypted)
00860             {
00861                 throw new DocumentLockedException("A password is necessary to render the document!");
00862             }
00863 
00864             return GetRenderedSize(Pages[pageNumber].Bounds, zoom, pixelFormat);
00865         }
00866 
00867 /// <summary>
00868 /// Determine how many bytes will be necessary to render the specified region in page units at the
      specified zoom level, using the the specified pixel format.
00869 /// </summary>
00870 /// <param name="region">The region that will be rendered.</param>
00871 /// <param name="zoom">The scale at which the region will be rendered.  This will determine the size
      in pixel of the image.</param>
00872 /// <param name="pixelFormat">The format of the pixels data.</param>
00873 /// <returns>An integer representing the number of bytes that will be necessary to store the pixel
      data of the rendered image.</returns>
00874         public static int GetRenderedSize(Rectangle region, double zoom, PixelFormats pixelFormat)
00875         {
00876             float x0 = region.X0 * (float)zoom;
00877             float y0 = region.Y0 * (float)zoom;
00878             float x1 = region.X1 * (float)zoom;
00879             float y1 = region.Y1 * (float)zoom;
00880 
00881             Rectangle scaledRect = new Rectangle(x0, y0, x1, y1);
00882             RoundedRectangle bounds = scaledRect.Round();
00883 
00884             int width = bounds.Width;
00885             int height = bounds.Height;
00886 
00887             switch (pixelFormat)
00888             {
00889                 case PixelFormats.RGB:
00890                 case PixelFormats.BGR:
00891                     return width * height * 3;
00892                 case PixelFormats.RGBA:
00893                 case PixelFormats.BGRA:
00894                     return width * height * 4;
00895             }
00896 
00897             return -1;
00898         }
00899 
00900 /// <summary>
00901 /// Save (part of) a page to an image file in the specified format.
00902 /// </summary>
00903 /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
00904 /// <param name="region">The region of the page to render in page units.</param>
00905 /// <param name="zoom">The scale at which the page will be rendered.  This will determine the size in
      pixel of the image.</param>
00906 /// <param name="pixelFormat">The format of the pixel data.</param>
00907 /// <param name="fileName">The path to the output file.</param>
00908 /// <param name="fileType">The output format of the file.</param>
00909 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included in the display list that is generated.  Otherwise, only the page contents are
      included.</param>
00910         public void SaveImage(int pageNumber, Rectangle region, double zoom, PixelFormats pixelFormat,
      string fileName, RasterOutputFileTypes fileType, bool includeAnnotations = true)
00911         {
00912             if (this.EncryptionState == EncryptionState.Encrypted)
00913             {
00914                 throw new DocumentLockedException("A password is necessary to render the document!");
00915             }
00916 
00917             if (pixelFormat == PixelFormats.RGBA && fileType == RasterOutputFileTypes.PNM)
00918             {
00919                 throw new ArgumentException("Cannot save an image with alpha channel in PNM format!",
      nameof(fileType));
00920             }
00921 
00922             if (pixelFormat != PixelFormats.RGB && fileType == RasterOutputFileTypes.JPEG)
00923             {
00924                 throw new ArgumentException("The JPEG format only supports RGB pixel data without an
      alpha channel!", nameof(fileType));
00925             }
00926 
00927             if ((pixelFormat != PixelFormats.RGB && pixelFormat != PixelFormats.RGBA) && fileType ==
```

```
      RasterOutputFileTypes.PNG)
00928             {
00929                 throw new ArgumentException("The PNG format only supports RGB or RGBA pixel data!",
      nameof(fileType));
00930             }
00931
00932             if (DisplayLists[pageNumber] == null)
00933             {
00934                 DisplayLists[pageNumber] = new MuPDFDisplayList(this.OwnerContext,
      this.Pages[pageNumber], includeAnnotations);
00935             }
00936
00937             if (zoom < 0.000001 | zoom * region.Width <= 0.001 || zoom * region.Height <= 0.001)
00938             {
00939                 throw new ArgumentOutOfRangeException(nameof(zoom), zoom, "The zoom factor is too
      small!");
00940             }
00941
00942             if (this.ImageXRes != 72 || this.ImageYRes != 72)
00943             {
00944                 zoom *= Math.Sqrt(this.ImageXRes * this.ImageYRes) / 72;
00945                 region = new Rectangle(region.X0 * 72 / this.ImageXRes, region.Y0 * 72 /
      this.ImageYRes, region.X1 * 72 / this.ImageXRes, region.Y1 * 72 / this.ImageYRes);
00946             }
00947
00948             float fzoom = (float)zoom;
00949
00950             ExitCodes result;
00951
00952             using (UTF8EncodedString encodedFileName = new UTF8EncodedString(fileName))
00953             {
00954                 result = (ExitCodes)NativeMethods.SaveImage(OwnerContext.NativeContext,
      DisplayLists[pageNumber].NativeDisplayList, region.X0, region.Y0, region.X1, region.Y1, fzoom,
      (int)pixelFormat, encodedFileName.Address, (int)fileType, 90);
00955             }
00956
00957             switch (result)
00958             {
00959                 case ExitCodes.EXIT_SUCCESS:
00960                     break;
00961                 case ExitCodes.ERR_CANNOT_RENDER:
00962                     throw new MuPDFException("Cannot render page", result);
00963                 case ExitCodes.ERR_CANNOT_SAVE:
00964                     throw new MuPDFException("Cannot save to the output file", result);
00965                 default:
00966                     throw new MuPDFException("Unknown error", result);
00967             }
00968         }
00969
00970 /// <summary>
00971 /// Save (part of) a page to an image file in JPEG format, with the specified quality.
00972 /// </summary>
00973 /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
00974 /// <param name="region">The region of the page to render in page units.</param>
00975 /// <param name="zoom">The scale at which the page will be rendered.  This will determine the size in
      pixel of the image.</param>
00976 /// <param name="fileName">The path to the output file.</param>
00977 /// <param name="quality">The quality of the JPEG output file (ranging from 0 to 100).</param>
00978 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included in the display list that is generated.  Otherwise, only the page contents are
      included.</param>
00979         public void SaveImageAsJPEG(int pageNumber, Rectangle region, double zoom, string fileName,
      int quality, bool includeAnnotations = true)
00980         {
00981             if (this.EncryptionState == EncryptionState.Encrypted)
00982             {
00983                 throw new DocumentLockedException("A password is necessary to render the document!");
00984             }
00985
00986             if (quality < 0 || quality > 100)
00987             {
00988                 throw new ArgumentOutOfRangeException(nameof(quality), quality, "The JPEG quality must
      range between 0 and 100 (inclusive)!");
00989             }
00990
00991             if (DisplayLists[pageNumber] == null)
00992             {
00993                 DisplayLists[pageNumber] = new MuPDFDisplayList(this.OwnerContext,
      this.Pages[pageNumber], includeAnnotations);
00994             }
00995
00996             if (zoom < 0.000001 | zoom * region.Width <= 0.001 || zoom * region.Height <= 0.001)
00997             {
00998                 throw new ArgumentOutOfRangeException(nameof(zoom), zoom, "The zoom factor is too
      small!");
00999             }
01000
```

```
01001             if (this.ImageXRes != 72 || this.ImageYRes != 72)
01002             {
01003                 zoom *= Math.Sqrt(this.ImageXRes * this.ImageYRes) / 72;
01004                 region = new Rectangle(region.X0 * 72 / this.ImageXRes, region.Y0 * 72 /
      this.ImageYRes, region.X1 * 72 / this.ImageXRes, region.Y1 * 72 / this.ImageYRes);
01005             }
01006
01007             float fzoom = (float)zoom;
01008
01009             ExitCodes result;
01010
01011             using (UTF8EncodedString encodedFileName = new UTF8EncodedString(fileName))
01012             {
01013                 result = (ExitCodes)NativeMethods.SaveImage(OwnerContext.NativeContext,
      DisplayLists[pageNumber].NativeDisplayList, region.X0, region.Y0, region.X1, region.Y1, fzoom,
      (int)PixelFormats.RGB, encodedFileName.Address, (int)RasterOutputFileTypes.JPEG, quality);
01014             }
01015
01016             switch (result)
01017             {
01018                 case ExitCodes.EXIT_SUCCESS:
01019                     break;
01020                 case ExitCodes.ERR_CANNOT_RENDER:
01021                     throw new MuPDFException("Cannot render page", result);
01022                 case ExitCodes.ERR_CANNOT_SAVE:
01023                     throw new MuPDFException("Cannot save to the output file", result);
01024                 default:
01025                     throw new MuPDFException("Unknown error", result);
01026             }
01027         }
01028
01029 /// <summary>
01030 /// Save a page to an image file in the specified format.
01031 /// </summary>
01032 /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
01033 /// <param name="zoom">The scale at which the page will be rendered.  This will determine the size in
      pixel of the image.</param>
01034 /// <param name="pixelFormat">The format of the pixel data.</param>
01035 /// <param name="fileName">The path to the output file.</param>
01036 /// <param name="fileType">The output format of the file.</param>
01037 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included in the display list that is generated.  Otherwise, only the page contents are
      included.</param>
01038         public void SaveImage(int pageNumber, double zoom, PixelFormats pixelFormat, string fileName,
      RasterOutputFileTypes fileType, bool includeAnnotations = true)
01039         {
01040             if (this.EncryptionState == EncryptionState.Encrypted)
01041             {
01042                 throw new DocumentLockedException("A password is necessary to render the document!");
01043             }
01044
01045             Rectangle region = this.Pages[pageNumber].Bounds;
01046             SaveImage(pageNumber, region, zoom, pixelFormat, fileName, fileType, includeAnnotations);
01047         }
01048
01049 /// <summary>
01050 /// Save a page to an image file in JPEG format, with the specified quality.
01051 /// </summary>
01052 /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
01053 /// <param name="zoom">The scale at which the page will be rendered.  This will determine the size in
      pixel of the image.</param>
01054 /// <param name="fileName">The path to the output file.</param>
01055 /// <param name="quality">The quality of the JPEG output file (ranging from 0 to 100).</param>
01056 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included in the display list that is generated.  Otherwise, only the page contents are
      included.</param>
01057         public void SaveImageAsJPEG(int pageNumber, double zoom, string fileName, int quality, bool
      includeAnnotations = true)
01058         {
01059             if (this.EncryptionState == EncryptionState.Encrypted)
01060             {
01061                 throw new DocumentLockedException("A password is necessary to render the document!");
01062             }
01063
01064             Rectangle region = this.Pages[pageNumber].Bounds;
01065             SaveImageAsJPEG(pageNumber, region, zoom, fileName, quality, includeAnnotations);
01066         }
01067
01068 /// <summary>
01069 /// Write (part of) a page to an image stream in the specified format.
01070 /// </summary>
01071 /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
01072 /// <param name="region">The region of the page to render in page units.</param>
01073 /// <param name="zoom">The scale at which the page will be rendered.  This will determine the size in
      pixel of the image.</param>
01074 /// <param name="pixelFormat">The format of the pixel data.</param>
01075 /// <param name="outputStream">The stream to which the image data will be written.</param>
```

```
01076 /// <param name="fileType">The output format of the image.</param>
01077 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included in the display list that is generated.  Otherwise, only the page contents are
      included.</param>
01078         public void WriteImage(int pageNumber, Rectangle region, double zoom, PixelFormats
      pixelFormat, Stream outputStream, RasterOutputFileTypes fileType, bool includeAnnotations = true)
01079         {
01080             if (this.EncryptionState == EncryptionState.Encrypted)
01081             {
01082                 throw new DocumentLockedException("A password is necessary to render the document!");
01083             }
01084 
01085             if (pixelFormat == PixelFormats.RGBA && fileType == RasterOutputFileTypes.PNM)
01086             {
01087                 throw new ArgumentException("Cannot save an image with alpha channel in PNM format!",
      nameof(fileType));
01088             }
01089 
01090             if (pixelFormat != PixelFormats.RGB && fileType == RasterOutputFileTypes.JPEG)
01091             {
01092                 throw new ArgumentException("The JPEG format only supports RGB pixel data without an
      alpha channel!", nameof(fileType));
01093             }
01094 
01095             if ((pixelFormat != PixelFormats.RGB && pixelFormat != PixelFormats.RGBA) && fileType ==
      RasterOutputFileTypes.PNG)
01096             {
01097                 throw new ArgumentException("The PNG format only supports RGB or RGBA pixel data!",
      nameof(fileType));
01098             }
01099 
01100             if (DisplayLists[pageNumber] == null)
01101             {
01102                 DisplayLists[pageNumber] = new MuPDFDisplayList(this.OwnerContext,
      this.Pages[pageNumber], includeAnnotations);
01103             }
01104 
01105             if (zoom < 0.000001 | zoom * region.Width <= 0.001 || zoom * region.Height <= 0.001)
01106             {
01107                 throw new ArgumentOutOfRangeException(nameof(zoom), zoom, "The zoom factor is too
      small!");
01108             }
01109 
01110             if (this.ImageXRes != 72 || this.ImageYRes != 72)
01111             {
01112                 zoom *= Math.Sqrt(this.ImageXRes * this.ImageYRes) / 72;
01113                 region = new Rectangle(region.X0 * 72 / this.ImageXRes, region.Y0 * 72 /
      this.ImageYRes, region.X1 * 72 / this.ImageXRes, region.Y1 * 72 / this.ImageYRes);
01114             }
01115 
01116             float fzoom = (float)zoom;
01117 
01118             IntPtr outputBuffer = IntPtr.Zero;
01119             IntPtr outputData = IntPtr.Zero;
01120             ulong outputDataLength = 0;
01121 
01122             ExitCodes result = (ExitCodes)NativeMethods.WriteImage(OwnerContext.NativeContext,
      DisplayLists[pageNumber].NativeDisplayList, region.X0, region.Y0, region.X1, region.Y1, fzoom,
      (int)pixelFormat, (int)fileType, 90, ref outputBuffer, ref outputData, ref outputDataLength);
01123 
01124             switch (result)
01125             {
01126                 case ExitCodes.EXIT_SUCCESS:
01127                     break;
01128                 case ExitCodes.ERR_CANNOT_RENDER:
01129                     throw new MuPDFException("Cannot render page", result);
01130                 case ExitCodes.ERR_CANNOT_CREATE_CONTEXT:
01131                     throw new MuPDFException("Cannot create the output buffer", result);
01132                 default:
01133                     throw new MuPDFException("Unknown error", result);
01134             }
01135 
01136             byte[] buffer = new byte[1024];
01137 
01138             while (outputDataLength > 0)
01139             {
01140                 int bytesToCopy = (int)Math.Min(buffer.Length, (long)outputDataLength);
01141                 Marshal.Copy(outputData, buffer, 0, bytesToCopy);
01142                 outputData = IntPtr.Add(outputData, bytesToCopy);
01143                 outputStream.Write(buffer, 0, bytesToCopy);
01144                 outputDataLength -= (ulong)bytesToCopy;
01145             }
01146 
01147             NativeMethods.DisposeBuffer(OwnerContext.NativeContext, outputBuffer);
01148         }
01149 
01150 
```

```
01151 /// <summary>
01152 /// Write (part of) a page to an image stream in JPEG format, with the specified quality.
01153 /// </summary>
01154 /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
01155 /// <param name="region">The region of the page to render in page units.</param>
01156 /// <param name="zoom">The scale at which the page will be rendered.  This will determine the size in
      pixel of the image.</param>
01157 /// <param name="outputStream">The stream to which the image data will be written.</param>
01158 /// <param name="quality">The quality of the JPEG output (ranging from 0 to 100).</param>
01159 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included in the display list that is generated.  Otherwise, only the page contents are
      included.</param>
01160         public void WriteImageAsJPEG(int pageNumber, Rectangle region, double zoom, Stream
      outputStream, int quality, bool includeAnnotations = true)
01161         {
01162             if (this.EncryptionState == EncryptionState.Encrypted)
01163             {
01164                 throw new DocumentLockedException("A password is necessary to render the document!");
01165             }
01166 
01167             if (quality < 0 || quality > 100)
01168             {
01169                 throw new ArgumentOutOfRangeException(nameof(quality), quality, "The JPEG quality must
      range between 0 and 100 (inclusive)!");
01170             }
01171 
01172             if (DisplayLists[pageNumber] == null)
01173             {
01174                 DisplayLists[pageNumber] = new MuPDFDisplayList(this.OwnerContext,
      this.Pages[pageNumber], includeAnnotations);
01175             }
01176 
01177             if (zoom < 0.000001 | zoom * region.Width <= 0.001 || zoom * region.Height <= 0.001)
01178             {
01179                 throw new ArgumentOutOfRangeException(nameof(zoom), zoom, "The zoom factor is too
      small!");
01180             }
01181 
01182             if (this.ImageXRes != 72 || this.ImageYRes != 72)
01183             {
01184                 zoom *= Math.Sqrt(this.ImageXRes * this.ImageYRes) / 72;
01185                 region = new Rectangle(region.X0 * 72 / this.ImageXRes, region.Y0 * 72 /
      this.ImageYRes, region.X1 * 72 / this.ImageXRes, region.Y1 * 72 / this.ImageYRes);
01186             }
01187 
01188             float fzoom = (float)zoom;
01189 
01190             IntPtr outputBuffer = IntPtr.Zero;
01191             IntPtr outputData = IntPtr.Zero;
01192             ulong outputDataLength = 0;
01193 
01194             ExitCodes result = (ExitCodes)NativeMethods.WriteImage(OwnerContext.NativeContext,
      DisplayLists[pageNumber].NativeDisplayList, region.X0, region.Y0, region.X1, region.Y1, fzoom,
      (int)PixelFormats.RGB, (int)RasterOutputFileTypes.JPEG, quality, ref outputBuffer, ref outputData, ref
      outputDataLength);
01195 
01196             switch (result)
01197             {
01198                 case ExitCodes.EXIT_SUCCESS:
01199                     break;
01200                 case ExitCodes.ERR_CANNOT_RENDER:
01201                     throw new MuPDFException("Cannot render page", result);
01202                 case ExitCodes.ERR_CANNOT_CREATE_CONTEXT:
01203                     throw new MuPDFException("Cannot create the output buffer", result);
01204                 default:
01205                     throw new MuPDFException("Unknown error", result);
01206             }
01207 
01208             byte[] buffer = new byte[1024];
01209 
01210             while (outputDataLength > 0)
01211             {
01212                 int bytesToCopy = (int)Math.Min(buffer.Length, (long)outputDataLength);
01213                 Marshal.Copy(outputData, buffer, 0, bytesToCopy);
01214                 outputData = IntPtr.Add(outputData, bytesToCopy);
01215                 outputStream.Write(buffer, 0, bytesToCopy);
01216                 outputDataLength -= (ulong)bytesToCopy;
01217             }
01218 
01219             NativeMethods.DisposeBuffer(OwnerContext.NativeContext, outputBuffer);
01220         }
01221 
01222 /// <summary>
01223 /// Write a page to an image stream in the specified format.
01224 /// </summary>
01225 /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
01226 /// <param name="zoom">The scale at which the page will be rendered.  This will determine the size in
```

```
pixel of the image.</param>
01227 /// <param name="pixelFormat">The format of the pixel data.</param>
01228 /// <param name="outputStream">The stream to which the image data will be written.</param>
01229 /// <param name="fileType">The output format of the image.</param>
01230 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
signatures) are included in the display list that is generated.  Otherwise, only the page contents are
included.</param>
01231         public void WriteImage(int pageNumber, double zoom, PixelFormats pixelFormat, Stream
outputStream, RasterOutputFileTypes fileType, bool includeAnnotations = true)
01232         {
01233             if (this.EncryptionState == EncryptionState.Encrypted)
01234             {
01235                 throw new DocumentLockedException("A password is necessary to render the document!");
01236             }
01237 
01238             Rectangle region = this.Pages[pageNumber].Bounds;
01239             WriteImage(pageNumber, region, zoom, pixelFormat, outputStream, fileType,
includeAnnotations);
01240         }
01241 
01242 /// <summary>
01243 /// Write a page to an image stream in JPEG format, with the specified quality.
01244 /// </summary>
01245 /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
01246 /// <param name="zoom">The scale at which the page will be rendered.  This will determine the size in
pixel of the image.</param>
01247 /// <param name="outputStream">The stream to which the image data will be written.</param>
01248 /// <param name="quality">The quality of the JPEG output (ranging from 0 to 100).</param>
01249 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
signatures) are included in the display list that is generated.  Otherwise, only the page contents are
included.</param>
01250         public void WriteImageAsJPEG(int pageNumber, double zoom, Stream outputStream, int quality,
bool includeAnnotations = true)
01251         {
01252             if (this.EncryptionState == EncryptionState.Encrypted)
01253             {
01254                 throw new DocumentLockedException("A password is necessary to render the document!");
01255             }
01256 
01257             Rectangle region = this.Pages[pageNumber].Bounds;
01258             WriteImageAsJPEG(pageNumber, region, zoom, outputStream, quality, includeAnnotations);
01259         }
01260 
01261 /// <summary>
01262 /// Create a new document containing the specified (parts of) pages from other documents.
01263 /// </summary>
01264 /// <param name="context">The context that was used to open the documents.</param>
01265 /// <param name="fileName">The output file name.</param>
01266 /// <param name="fileType">The output file format.</param>
01267 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
signatures) are included in the display list that is generated.  Otherwise, only the page contents are
included.</param>
01268 /// <param name="pages">The pages to include in the document.  The "page" element specifies the page,
the "region" element the area of the page that should be included in the document, and the "zoom"
element how much the region should be scaled.</param>
01269         public static void CreateDocument(MuPDFContext context, string fileName,
DocumentOutputFileTypes fileType, bool includeAnnotations = true, params (MuPDFPage page, Rectangle
region, float zoom)[] pages)
01270         {
01271             if (fileType == DocumentOutputFileTypes.SVG && pages.Length > 1)
01272             {
01273                 //Actually, you can, but the library creates multiple files appending numbers after
each name (e.g.  page1.svg, page2.svg, ...), which is ugly and may have unintended consequences.
01274                 //If you really want to do this, you can call this method multiple times.
01275                 throw new ArgumentException("You cannot create an SVG document with more than one
page!", nameof(pages));
01276             }
01277 
01278             string originalFileName = fileName;
01279 
01280             if (fileType == DocumentOutputFileTypes.SVG)
01281             {
01282                 //For SVG documents, the library annoyingly alters the output file name, appending a
"1" just before the extension (e.g.  document.svg -> document1.svg).  Since users may not be expecting
this, it is best to render to a temporary file and then move it to the specified location.
01283                 fileName = Path.GetTempFileName();
01284             }
01285 
01286             IntPtr documentWriter = IntPtr.Zero;
01287 
01288             ExitCodes result;
01289 
01290             // Encode the file name in UTF-8 in unmanaged memory.
01291             using (UTF8EncodedString encodedFileName = new UTF8EncodedString(fileName))
01292             {
01293                 //Initialise document writer.
01294                 result = (ExitCodes)NativeMethods.CreateDocumentWriter(context.NativeContext,
```

```
    encodedFileName.Address, (int)fileType, ref documentWriter);
01295             }
01296
01297             switch (result)
01298             {
01299                 case ExitCodes.EXIT_SUCCESS:
01300                     break;
01301                 case ExitCodes.ERR_CANNOT_CREATE_WRITER:
01302                     throw new MuPDFException("Cannot create the document writer", result);
01303                 default:
01304                     throw new MuPDFException("Unknown error", result);
01305             }
01306
01307             //Write pages.
01308             for (int i = 0; i < pages.Length; i++)
01309             {
01310                 MuPDFDocument doc = pages[i].page.OwnerDocument;
01311                 int pageNum = pages[i].page.PageNumber;
01312
01313                 if (doc.DisplayLists[pageNum] == null)
01314                 {
01315                     doc.DisplayLists[pageNum] = new MuPDFDisplayList(doc.OwnerContext,
    doc.Pages[pageNum], includeAnnotations);
01316                 }
01317
01318                 Rectangle region = pages[i].region;
01319                 double zoom = pages[i].zoom;
01320
01321                 if (pages[i].page.OwnerDocument.ImageXRes != 72 ||
    pages[i].page.OwnerDocument.ImageYRes != 72)
01322                 {
01323                     zoom *= Math.Sqrt(pages[i].page.OwnerDocument.ImageXRes *
    pages[i].page.OwnerDocument.ImageYRes) / 72;
01324                     region = new Rectangle(region.X0 * 72 / pages[i].page.OwnerDocument.ImageXRes,
    region.Y0 * 72 / pages[i].page.OwnerDocument.ImageYRes, region.X1 * 72 /
    pages[i].page.OwnerDocument.ImageXRes, region.Y1 * 72 / pages[i].page.OwnerDocument.ImageYRes);
01325                 }
01326
01327                 result = (ExitCodes)NativeMethods.WriteSubDisplayListAsPage(context.NativeContext,
    doc.DisplayLists[pageNum].NativeDisplayList, region.X0, region.Y0, region.X1, region.Y1, (float)zoom,
    documentWriter);
01328
01329                 switch (result)
01330                 {
01331                     case ExitCodes.EXIT_SUCCESS:
01332                         break;
01333                     case ExitCodes.ERR_CANNOT_RENDER:
01334                         throw new MuPDFException("Cannot render page " + i.ToString(), result);
01335                     default:
01336                         throw new MuPDFException("Unknown error", result);
01337                 }
01338             }
01339
01340             //Close and dispose the document writer.
01341             result = (ExitCodes)NativeMethods.FinalizeDocumentWriter(context.NativeContext,
    documentWriter);
01342
01343             switch (result)
01344             {
01345                 case ExitCodes.EXIT_SUCCESS:
01346                     break;
01347                 case ExitCodes.ERR_CANNOT_CLOSE_DOCUMENT:
01348                     throw new MuPDFException("Cannot finalise the document", result);
01349                 default:
01350                     throw new MuPDFException("Unknown error", result);
01351             }
01352
01353             if (fileType == DocumentOutputFileTypes.SVG)
01354             {
01355                 //Move the temporary file to the location specified by the user.
01356                 //The library has altered the temporary file name by appending a "1" before the
    extension.
01357                 string tempFileName = Path.Combine(Path.GetDirectoryName(fileName),
    Path.GetFileNameWithoutExtension(fileName) + "1" + Path.GetExtension(fileName));
01358
01359                 //Overwrite existing file.
01360                 if (File.Exists(originalFileName))
01361                 {
01362                     File.Delete(originalFileName);
01363                 }
01364
01365                 File.Move(tempFileName, originalFileName);
01366             }
01367         }
01368
01369 /// <summary>
01370 /// Create a new document containing the specified pages from other documents.
```

```
01371 /// </summary>
01372 /// <param name="context">The context that was used to open the documents.</param>
01373 /// <param name="fileName">The output file name.</param>
01374 /// <param name="fileType">The output file format.</param>
01375 /// <param name="pages">The pages to include in the document.</param>
01376 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included in the display list that is generated.  Otherwise, only the page contents are
      included.</param>
01377         public static void CreateDocument(MuPDFContext context, string fileName,
      DocumentOutputFileTypes fileType, bool includeAnnotations = true, params MuPDFPage[] pages)
01378         {
01379             (MuPDFPage, Rectangle, float)[] boundedPages = new (MuPDFPage, Rectangle,
      float)[pages.Length];
01380 
01381             for (int i = 0; i < pages.Length; i++)
01382             {
01383                 boundedPages[i] = (pages[i], pages[i].Bounds, 1);
01384             }
01385 
01386             CreateDocument(context, fileName, fileType, includeAnnotations, boundedPages);
01387         }
01388 
01389 
01390 /// <summary>
01391 /// Creates a new <see cref="MuPDFStructuredTextPage"/> from the specified page.  This contains
      information about the text layout that can be used for highlighting and searching.  The reading order
      is taken from the order the text is drawn in the source file, so may not be accurate.
01392 /// </summary>
01393 /// <param name="pageNumber">The number of the page (starting at 0)</param>
01394 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included.  Otherwise, only the page contents are included.</param>
01395 /// <returns>A <see cref="MuPDFStructuredTextPage"/> containing a structured text representation of
      the page.</returns>
01396         public MuPDFStructuredTextPage GetStructuredTextPage(int pageNumber, bool includeAnnotations =
      true)
01397         {
01398             if (this.EncryptionState == EncryptionState.Encrypted)
01399             {
01400                 throw new DocumentLockedException("A password is necessary to render the document!");
01401             }
01402 
01403             if (DisplayLists[pageNumber] == null)
01404             {
01405                 DisplayLists[pageNumber] = new MuPDFDisplayList(this.OwnerContext,
      this.Pages[pageNumber], includeAnnotations);
01406             }
01407 
01408             return new MuPDFStructuredTextPage(this.OwnerContext, this.DisplayLists[pageNumber], null,
      1, new Rectangle());
01409         }
01410 
01411 /// <summary>
01412 /// Creates a new <see cref="MuPDFStructuredTextPage"/> from the specified page, using optical
      character recognition (OCR) to determine what text is written on the image.  This contains information
      about the text layout that can be used for highlighting and searching.
01413 /// </summary>
01414 /// <param name="pageNumber">The number of the page (starting at 0)</param>
01415 /// <param name="ocrLanguage">The language to use for optical character recognition (OCR). If this is
      null, no OCR is performed.</param>
01416 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included.  Otherwise, only the page contents are included.</param>
01417 /// <param name="cancellationToken">A <see cref="CancellationToken"/> used to cancel the operation.
      Providing a value other than the default is not supported on Windows x86 and will throw a runtime
      exception.</param>
01418 /// <param name="progress">An <see cref="IProgress{OCRProgressInfo}"/> used to report progress.
      Providing a value other than null is not supported on Windows x86 and will throw a runtime
      exception.</param>
01419 /// <returns>A <see cref="MuPDFStructuredTextPage"/> containing a structured text representation of
      the page.</returns>
01420         public MuPDFStructuredTextPage GetStructuredTextPage(int pageNumber, TesseractLanguage
      ocrLanguage, bool includeAnnotations = true, CancellationToken cancellationToken = default,
      IProgress<OCRProgressInfo> progress = null)
01421         {
01422             if (this.EncryptionState == EncryptionState.Encrypted)
01423             {
01424                 throw new DocumentLockedException("A password is necessary to render the document!");
01425             }
01426 
01427             if (DisplayLists[pageNumber] == null)
01428             {
01429                 DisplayLists[pageNumber] = new MuPDFDisplayList(this.OwnerContext,
      this.Pages[pageNumber], includeAnnotations);
01430             }
01431 
01432             double zoom = 1;
01433             Rectangle region = this.Pages[pageNumber].Bounds;
01434 
```

```
01435             if (this.ImageXRes != 72 || this.ImageYRes != 72)
01436             {
01437                 zoom *= Math.Sqrt(this.ImageXRes * this.ImageYRes) / 72;
01438                 region = new Rectangle(region.X0 * 72 / this.ImageXRes, region.Y0 * 72 /
      this.ImageYRes, region.X1 * 72 / this.ImageXRes, region.Y1 * 72 / this.ImageYRes);
01439             }
01440 
01441             return new MuPDFStructuredTextPage(this.OwnerContext, this.DisplayLists[pageNumber],
      ocrLanguage, zoom, region, cancellationToken, progress);
01442         }
01443 
01444 /// <summary>
01445 /// Creates a new <see cref="MuPDFStructuredTextPage"/> from the specified page, using optical
      character recognition (OCR) to determine what text is written on the image.  This contains information
      about the text layout that can be used for highlighting and searching.  The OCR step is run
      asynchronously, e.g.  to avoid blocking the UI thread.
01446 /// </summary>
01447 /// <param name="pageNumber">The number of the page (starting at 0)</param>
01448 /// <param name="ocrLanguage">The language to use for optical character recognition (OCR). If this is
      null, no OCR is performed.</param>
01449 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included.  Otherwise, only the page contents are included.</param>
01450 /// <param name="cancellationToken">A <see cref="CancellationToken"/> used to cancel the operation.
      Providing a value other than the default is not supported on Windows x86 and will throw a runtime
      exception.</param>
01451 /// <param name="progress">An <see cref="IProgress{OCRProgressInfo}"/> used to report progress.
      Providing a value other than null is not supported on Windows x86 and will throw a runtime
      exception.</param>
01452 /// <returns>A <see cref="MuPDFStructuredTextPage"/> containing a structured text representation of
      the page.</returns>
01453         public async Task<MuPDFStructuredTextPage> GetStructuredTextPageAsync(int pageNumber,
      TesseractLanguage ocrLanguage, bool includeAnnotations = true, CancellationToken cancellationToken =
      default, IProgress<OCRProgressInfo> progress = null)
01454         {
01455             if (this.EncryptionState == EncryptionState.Encrypted)
01456             {
01457                 throw new DocumentLockedException("A password is necessary to render the document!");
01458             }
01459 
01460             if (DisplayLists[pageNumber] == null)
01461             {
01462                 DisplayLists[pageNumber] = new MuPDFDisplayList(this.OwnerContext,
      this.Pages[pageNumber], includeAnnotations);
01463             }
01464 
01465             double zoom = 1;
01466             Rectangle region = this.Pages[pageNumber].Bounds;
01467 
01468             if (this.ImageXRes != 72 || this.ImageYRes != 72)
01469             {
01470                 zoom *= Math.Sqrt(this.ImageXRes * this.ImageYRes) / 72;
01471                 region = new Rectangle(region.X0 * 72 / this.ImageXRes, region.Y0 * 72 /
      this.ImageYRes, region.X1 * 72 / this.ImageXRes, region.Y1 * 72 / this.ImageYRes);
01472             }
01473 
01474             return await Task.Run(() => new MuPDFStructuredTextPage(this.OwnerContext,
      this.DisplayLists[pageNumber], ocrLanguage, zoom, region, cancellationToken, progress));
01475         }
01476 
01477 /// <summary>
01478 /// Extracts all the text from the document and returns it as a <see cref="string"/>.  The reading
      order is taken from the order the text is drawn in the source file, so may not be accurate.
01479 /// </summary>
01480 /// <param name="separator">The character(s) used to separate the text lines obtained from the
      document.  If this is <see langword="null" />, <see cref="Environment.NewLine"/> is used as a default
      separator.</param>
01481 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included.  Otherwise, only the page contents are included.</param>
01482 /// <returns>A <see cref="string"/> containing all the text in the document.  Characters are converted
      from the UTF-8 representation used in the document to equivalent UTF-16 <see
      cref="string"/>s.</returns>
01483         public string ExtractText(string separator = null, bool includeAnnotations = true)
01484         {
01485             if (this.EncryptionState == EncryptionState.Encrypted)
01486             {
01487                 throw new DocumentLockedException("A password is necessary to render the document!");
01488             }
01489 
01490             separator = separator ??  Environment.NewLine;
01491 
01492             var text = new StringBuilder();
01493             bool started = false;
01494 
01495             for (int i = 0; i < this.Pages.Count; i++)
01496             {
01497                 MuPDFStructuredTextPage structuredTextPage = this.GetStructuredTextPage(i,
      includeAnnotations);
```

```
01498                 foreach (MuPDFStructuredTextBlock textBlock in
      structuredTextPage.StructuredTextBlocks)
01499                 {
01500                     var numLines = textBlock.Count;
01501                     for (var j = 0; j < numLines; j++)
01502                     {
01503                         if (!string.IsNullOrWhiteSpace(textBlock[j].Text))
01504                         {
01505                             if (started)
01506                             {
01507                                 text.Append(separator);
01508                             }
01509                             else
01510                             {
01511                                 started = true;
01512                             }
01513
01514                             text.Append(textBlock[j].Text);
01515                         }
01516                     }
01517                 }
01518             }
01519
01520             return text.ToString();
01521         }
01522
01523 /// <summary>
01524 /// Extracts all the text from the document and returns it as a <see cref="string"/>, using optical
      character recognition (OCR) to determine what text is written on the image.
01525 /// </summary>
01526 /// <param name="separator">The character(s) used to separate the text lines obtained from the
      document.  If this is <see langword="null" />, <see cref="Environment.NewLine"/> is used as a default
      separator.</param>
01527 /// <param name="ocrLanguage">The language to use for optical character recognition (OCR). If this is
      null, no OCR is performed.</param>
01528 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included.  Otherwise, only the page contents are included.</param>
01529 /// <returns>A <see cref="string"/> containing all the text in the document.  Characters are converted
      from the UTF-8 representation used in the document to equivalent UTF-16 <see
      cref="string"/>s.</returns>
01530         public string ExtractText(TesseractLanguage ocrLanguage, string separator = null, bool
      includeAnnotations = true)
01531         {
01532             if (this.EncryptionState == EncryptionState.Encrypted)
01533             {
01534                 throw new DocumentLockedException("A password is necessary to render the document!");
01535             }
01536
01537             separator = separator ??  Environment.NewLine;
01538
01539             var text = new StringBuilder();
01540             bool started = false;
01541
01542             for (int i = 0; i < this.Pages.Count; i++)
01543             {
01544                 MuPDFStructuredTextPage structuredTextPage = this.GetStructuredTextPage(i,
      ocrLanguage, includeAnnotations);
01545                 foreach (MuPDFStructuredTextBlock textBlock in
      structuredTextPage.StructuredTextBlocks)
01546                 {
01547                     var numLines = textBlock.Count;
01548                     for (var j = 0; j < numLines; j++)
01549                     {
01550                         if (!string.IsNullOrWhiteSpace(textBlock[j].Text))
01551                         {
01552                             if (started)
01553                             {
01554                                 text.Append(separator);
01555                             }
01556                             else
01557                             {
01558                                 started = true;
01559                             }
01560
01561                             text.Append(textBlock[j].Text);
01562                         }
01563                     }
01564                 }
01565             }
01566
01567             return text.ToString();
01568         }
01569
01570 /// <summary>
01571 /// Extracts all the text from the document and returns it as a <see cref="string"/>, using optical
      character recognition (OCR) to determine what text is written on the image.  The OCR step is run
      asynchronously, e.g.  to avoid blocking the UI thread.
```

```
01572 /// </summary>
01573 /// <param name="separator">The character(s) used to separate the text lines obtained from the
      document.  If this is <see langword="null" />, <see cref="Environment.NewLine"/> is used as a default
      separator.</param>
01574 /// <param name="ocrLanguage">The language to use for optical character recognition (OCR). If this is
      null, no OCR is performed.</param>
01575 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
      signatures) are included.  Otherwise, only the page contents are included.</param>
01576 /// <param name="cancellationToken">A <see cref="CancellationToken"/> used to cancel the
      operation.</param>
01577 /// <param name="progress">An <see cref="IProgress{OCRProgressInfo}"/> used to report
      progress.</param>
01578 /// <returns>A <see cref="string"/> containing all the text in the document.  Characters are converted
      from the UTF-8 representation used in the document to equivalent UTF-16 <see
      cref="string"/>s.</returns>
01579         public async Task<string> ExtractTextAsync(TesseractLanguage ocrLanguage, string separator =
      null, bool includeAnnotations = true, CancellationToken cancellationToken = default,
      IProgress<OCRProgressInfo> progress = null)
01580         {
01581             if (this.EncryptionState == EncryptionState.Encrypted)
01582             {
01583                 throw new DocumentLockedException("A password is necessary to render the document!");
01584             }
01585 
01586             separator = separator ??  Environment.NewLine;
01587 
01588             var text = new StringBuilder();
01589             bool started = false;
01590 
01591             for (int i = 0; i < this.Pages.Count; i++)
01592             {
01593                 MuPDFStructuredTextPage structuredTextPage = await this.GetStructuredTextPageAsync(i,
      ocrLanguage, includeAnnotations, cancellationToken, progress);
01594                 foreach (MuPDFStructuredTextBlock textBlock in
      structuredTextPage.StructuredTextBlocks)
01595                 {
01596                     var numLines = textBlock.Count;
01597                     for (var j = 0; j < numLines; j++)
01598                     {
01599                         if (!string.IsNullOrWhiteSpace(textBlock[j].Text))
01600                         {
01601                             if (started)
01602                             {
01603                                 text.Append(separator);
01604                             }
01605                             else
01606                             {
01607                                 started = true;
01608                             }
01609 
01610                             text.Append(textBlock[j].Text);
01611                         }
01612                     }
01613                 }
01614             }
01615 
01616             return text.ToString();
01617         }
01618 
01619 /// <summary>
01620 /// Attempts to unlock the document with the supplied password.
01621 /// </summary>
01622 /// <param name="password">The user or owner password to use to unlock the document.</param>
01623 /// <returns><see langword="true"/>If the document was successfully unlocked (or if it was never
      locked to begin with), <see langword="false"/> if the password was incorrect and the document is still
      locked.</returns>
01624 /// <remarks>This method can be used both to unlock an encrypted document and to check whether the
      supplied owner password is correct.</remarks>
01625         public bool TryUnlock(string password)
01626         {
01627             return TryUnlock(password, out _);
01628         }
01629 
01630 /// <summary>
01631 /// Attempts to unlock the document with the supplied password.
01632 /// </summary>
01633 /// <param name="password">The user or owner password to use to unlock the document.</param>
01634 /// <param name="passwordType">If the method returns true, this can be used to determine whether the
      supplied password was the user password or the owner password.  If the method returns <see
      langword="false"/>,
01635 /// this can be used to determine whether a user password and/or an owner password are
      required.</param>
01636 /// <returns><see langword="true"/>If the document was successfully unlocked (or if it was never
      locked to begin with), <see langword="false"/> if the password was incorrect and the document is still
      locked.</returns>
01637 /// <remarks>This method can be used both to unlock an encrypted document and to check whether the
      supplied owner password is correct.</remarks>
```

```
01638         public bool TryUnlock(string password, out PasswordTypes passwordType)
01639         {
01640             int result = NativeMethods.UnlockWithPassword(this.OwnerContext.NativeContext,
      this.NativeDocument, password);
01641 
01642             int pt = 0;
01643 
01644             switch (result)
01645             {
01646                 case 0:
01647                     pt = 0;
01648 
01649                     if (this.EncryptionState == EncryptionState.Encrypted)
01650                     {
01651                         pt |= 1;
01652                     }
01653 
01654                     if (this.RestrictionState == RestrictionState.Restricted)
01655                     {
01656                         pt |= 2;
01657                     }
01658 
01659                     passwordType = (PasswordTypes)pt;
01660                     return !(this.EncryptionState == EncryptionState.Encrypted ||
      this.RestrictionState == RestrictionState.Restricted);
01661                 case 1:
01662                     passwordType = PasswordTypes.None;
01663                     return true;
01664                 case 2:
01665                     if (this.EncryptionState == EncryptionState.Encrypted)
01666                     {
01667                         this.EncryptionState = EncryptionState.Unlocked;
01668                     }
01669                     passwordType = PasswordTypes.User;
01670                     return true;
01671                 case 4:
01672                     if (this.RestrictionState == RestrictionState.Restricted)
01673                     {
01674                         this.RestrictionState = RestrictionState.Unlocked;
01675                     }
01676                     passwordType = PasswordTypes.Owner;
01677                     return true;
01678                 case 6:
01679                     pt = 0;
01680 
01681                     if (this.EncryptionState == EncryptionState.Encrypted)
01682                     {
01683                         this.EncryptionState = EncryptionState.Unlocked;
01684                         pt |= 1;
01685                     }
01686 
01687                     if (this.RestrictionState == RestrictionState.Restricted)
01688                     {
01689                         this.RestrictionState = RestrictionState.Unlocked;
01690                         pt |= 2;
01691                     }
01692                     passwordType = (PasswordTypes)pt;
01693                     return true;
01694                 default:
01695                     throw new ArgumentOutOfRangeException("Unexpected return value when unlocking the
      document:  " + result.ToString());
01696             }
01697         }
01698 
01699 
01700         private bool disposedValue;
01701 
01702 ///<inheritdoc/>
01703         protected virtual void Dispose(bool disposing)
01704         {
01705             if (!disposedValue)
01706             {
01707                 if (disposing)
01708                 {
01709                     Pages.Dispose();
01710                     foreach (MuPDFDisplayList list in DisplayLists)
01711                     {
01712                         list?.Dispose();
01713                     }
01714                     DataHandle?.Free();
01715                     DataHolder?.Dispose();
01716                 }
01717 
01718                 NativeMethods.DisposeDocument(OwnerContext.NativeContext, NativeDocument);
01719 
01720                 if (NativeStream != IntPtr.Zero)
01721                 {
```

```
01722                     NativeMethods.DisposeStream(OwnerContext.NativeContext, NativeStream);
01723                 }
01724
01725                 disposedValue = true;
01726             }
01727         }
01728
01729 ///<inheritdoc/>
01730         ~MuPDFDocument()
01731         {
01732             Dispose(disposing:  false);
01733         }
01734
01735 ///<inheritdoc/>
01736         public void Dispose()
01737         {
01738             Dispose(disposing:  true);
01739             GC.SuppressFinalize(this);
01740         }
01741     }
01742 }
```

8.8 MuPDFMultiThreadedPageRenderer.cs

```
00001 /*
00002 MuPDFCore - A set of multiplatform .NET Core bindings for MuPDF.
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 System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021 using System.Runtime.InteropServices;
00022 using System.Threading;
00023
00024 namespace MuPDFCore
00025 {
00026 /// <summary>
00027 /// A reusable thread to render part of an image.
00028 /// </summary>
00029     internal class RenderingThread :  IDisposable
00030     {
00031 /// <summary>
00032 /// A class to hold the data used internally by the rendering thread.
00033 /// </summary>
00034         private class RenderData
00035         {
00036             public IntPtr Context;
00037             public MuPDFDisplayList DisplayList;
00038             public Rectangle Region;
00039             public float Zoom;
00040             public IntPtr PixelStorage;
00041             public Rectangle PageBounds;
00042             public PixelFormats PixelFormat;
00043             public bool ClipToPageBounds;
00044         }
00045
00046 /// <summary>
00047 /// A lock object to prevent race conditions.
00048 /// </summary>
00049         private readonly object RenderDataLock;
00050
00051 /// <summary>
00052 /// The data used internally by the rendering thread.
00053 /// </summary>
00054         private readonly RenderData CurrentRenderData;
00055
00056 /// <summary>
00057 /// The actual thread that does the rendering.
00058 /// </summary>
00059         private readonly Thread Thread;
```

```
00060
00061 /// <summary>
00062 /// An <see cref="EventWaitHandle"/> that is set by the <see cref="Thread"/> when it finished
      rendering.
00063 /// </summary>
00064         private readonly EventWaitHandle SignalFromThread;
00065
00066 /// <summary>
00067 /// An <see cref="EventWaitHandle"/> that signals to the <see cref="Thread"/> that it should start
      rendering.
00068 /// </summary>
00069         private readonly EventWaitHandle SignalToThread;
00070
00071 /// <summary>
00072 /// An <see cref="EventWaitHandle"/> that signals that the object is being disposed and all activity
      should cease.
00073 /// </summary>
00074         private readonly EventWaitHandle DisposeSignal;
00075
00076 /// <summary>
00077 /// A pointer to a <see cref="Cookie"/> object that can be used to monitor the progress of the
      rendering or to abort it.
00078 /// </summary>
00079         private readonly IntPtr Cookie;
00080
00081 /// <summary>
00082 /// Performs the actual rendering.
00083 /// </summary>
00084         private void RenderAction()
00085         {
00086             ExitCodes result =
      (ExitCodes)NativeMethods.RenderSubDisplayList(this.CurrentRenderData.Context,
      this.CurrentRenderData.DisplayList.NativeDisplayList, this.CurrentRenderData.Region.X0,
      this.CurrentRenderData.Region.Y0, this.CurrentRenderData.Region.X1, this.CurrentRenderData.Region.Y1,
      this.CurrentRenderData.Zoom, (int)this.CurrentRenderData.PixelFormat,
      this.CurrentRenderData.PixelStorage, Cookie);
00087
00088             switch (result)
00089             {
00090                 case ExitCodes.EXIT_SUCCESS:
00091                     break;
00092                 case ExitCodes.ERR_CANNOT_RENDER:
00093                     throw new MuPDFException("Cannot render page", result);
00094                 default:
00095                     throw new MuPDFException("Unknown error", result);
00096             }
00097
00098             RoundedRectangle roundedRegion =
      this.CurrentRenderData.Region.Round(this.CurrentRenderData.Zoom);
00099             RoundedSize roundedSize = new RoundedSize(roundedRegion.Width, roundedRegion.Height);
00100
00101             if (this.CurrentRenderData.PixelFormat == PixelFormats.RGBA ||
      this.CurrentRenderData.PixelFormat == PixelFormats.BGRA)
00102             {
00103                 Utils.UnpremultiplyAlpha(this.CurrentRenderData.PixelStorage, roundedSize);
00104             }
00105
00106             if (this.CurrentRenderData.ClipToPageBounds &&
      !this.CurrentRenderData.PageBounds.Contains(this.CurrentRenderData.DisplayList.Bounds.Intersect(this.CurrentRenderData.
00107             {
00108                 Utils.ClipImage(this.CurrentRenderData.PixelStorage, roundedSize,
      this.CurrentRenderData.Region, this.CurrentRenderData.PageBounds, this.CurrentRenderData.PixelFormat);
00109             }
00110         }
00111
00112 /// <summary>
00113 /// Create a new <see cref="RenderingThread"/> instance.
00114 /// </summary>
00115         public RenderingThread()
00116         {
00117             //Initialize fields
00118             SignalFromThread = new EventWaitHandle(false, EventResetMode.ManualReset);
00119             SignalToThread = new EventWaitHandle(false, EventResetMode.ManualReset);
00120             DisposeSignal = new EventWaitHandle(false, EventResetMode.ManualReset);
00121
00122             CurrentRenderData = new RenderData();
00123             RenderDataLock = new object();
00124
00125             //Allocate unmanaged memory to hold the cookie.
00126             Cookie = Marshal.AllocHGlobal(Marshal.SizeOf<MuPDFCore.Cookie>());
00127
00128             //Initialise the rendering thread.
00129             this.Thread = new Thread(() =>
00130             {
00131                 EventWaitHandle[] handles = new EventWaitHandle[] { SignalToThread, DisposeSignal };
00132
00133                 while (true)
```

```
00134                 {
00135                     int handle = EventWaitHandle.WaitAny(handles);
00136 
00137                     if (handle == 0)
00138                     {
00139                         SignalToThread.Reset();
00140 
00141                         lock (RenderDataLock)
00142                         {
00143                             this.RenderAction();
00144                         }
00145 
00146                         SignalFromThread.Set();
00147                     }
00148                     else
00149                     {
00150                         SignalFromThread.Set();
00151                         break;
00152                     }
00153                 }
00154             });
00155 
00156             //Start the rendering thread.
00157             this.Thread.Start();
00158         }
00159 
00160 /// <summary>
00161 /// Start rendering a region of a display list to the specified destination.
00162 /// </summary>
00163 /// <param name="context">The rendering context.</param>
00164 /// <param name="displayList">The native display list that should be rendered.</param>
00165 /// <param name="region">The region that should be rendered.</param>
00166 /// <param name="zoom">The scale at which the region will be rendered.  This will determine the size
      in pixel of the image.</param>
00167 /// <param name="pixelStorage">The address of the buffer where the pixel data will be written.  There
      must be enough space available to write the values for all the pixels, otherwise this will fail
      catastrophically!</param>
00168 /// <param name="pixelFormat">The format of the pixel data.</param>
00169 /// <param name="pageBounds">The bounds of the page being rendererd.</param>
00170 /// <param name="clipToPageBounds">A boolean value indicating whether the rendered image should be
      clipped to the original page's bounds.  This can be relevant if the page has been "cropped" by
      altering its mediabox, but otherwise leaving the contents untouched.</param>
00171         public void Render(IntPtr context, MuPDFDisplayList displayList, Rectangle region, float zoom,
      IntPtr pixelStorage, PixelFormats pixelFormat, Rectangle pageBounds, bool clipToPageBounds)
00172         {
00173             lock (RenderDataLock)
00174             {
00175                 //Reset the cookie.
00176                 unsafe
00177                 {
00178                     Cookie* cookie = (Cookie*)Cookie;
00179 
00180                     cookie->abort = 0;
00181                     cookie->errors = 0;
00182                     cookie->incomplete = 0;
00183                     cookie->progress = 0;
00184                     cookie->progress_max = 0;
00185                 }
00186 
00187                 //Set up all the rendering data.
00188                 CurrentRenderData.Context = context;
00189                 CurrentRenderData.DisplayList = displayList;
00190                 CurrentRenderData.Region = region;
00191                 CurrentRenderData.Zoom = zoom;
00192                 CurrentRenderData.PixelStorage = pixelStorage;
00193                 CurrentRenderData.PixelFormat = pixelFormat;
00194                 CurrentRenderData.PageBounds = pageBounds;
00195                 CurrentRenderData.ClipToPageBounds = clipToPageBounds;
00196                 SignalToThread.Set();
00197             }
00198         }
00199 
00200 /// <summary>
00201 /// Wait until the current rendering operation finishes.
00202 /// </summary>
00203         public void WaitForRendering()
00204         {
00205             EventWaitHandle[] handles = new EventWaitHandle[] { SignalFromThread, DisposeSignal };
00206             int result = EventWaitHandle.WaitAny(handles);
00207             if (result == 0)
00208             {
00209                 SignalFromThread.Reset();
00210             }
00211         }
00212 
00213 /// <summary>
00214 /// Abort the current rendering operation.
```

```
00215 /// </summary>
00216         public void AbortRendering()
00217         {
00218             lock (RenderDataLock)
00219             {
00220                 unsafe
00221                 {
00222                     Cookie* cookie = (Cookie*)Cookie;
00223                     cookie->abort = 1;
00224                 }
00225             }
00226         }
00227 
00228 /// <summary>
00229 /// Get the progress of the current rendering operation.
00230 /// </summary>
00231 /// <returns>A <see cref="RenderProgress.ThreadRenderProgress"/> object containing the progress of the
      current rendering operation.</returns>
00232         public RenderProgress.ThreadRenderProgress GetProgress()
00233         {
00234             int progress;
00235             ulong maxProgress;
00236 
00237             unsafe
00238             {
00239                 Cookie* cookie = (Cookie*)Cookie;
00240 
00241                 progress = cookie->progress;
00242                 maxProgress = cookie->progress_max;
00243             }
00244 
00245             return new RenderProgress.ThreadRenderProgress(progress, maxProgress);
00246         }
00247 
00248         private bool disposedValue;
00249 
00250         protected virtual void Dispose(bool disposing)
00251         {
00252             if (!disposedValue)
00253             {
00254                 if (disposing)
00255                 {
00256                     DisposeSignal.Set();
00257                     Thread.Join();
00258                 }
00259 
00260                 Marshal.FreeHGlobal(Cookie);
00261 
00262                 disposedValue = true;
00263             }
00264         }
00265 
00266         public void Dispose()
00267         {
00268             Dispose(disposing:  true);
00269             GC.SuppressFinalize(this);
00270         }
00271     }
00272 
00273 /// <summary>
00274 /// A class that holds the necessary resources to render a page of a MuPDF document using multiple
      threads.
00275 /// </summary>
00276     public class MuPDFMultiThreadedPageRenderer :  IDisposable
00277     {
00278 /// <summary>
00279 /// The display list that is rendered by this renderer.
00280 /// </summary>
00281         private readonly MuPDFDisplayList DisplayList;
00282 
00283 /// <summary>
00284 /// The cloned contexts that are used by the <see cref="RenderingThreads"/> to render the display
      list.
00285 /// </summary>
00286         private readonly MuPDFContext[] Contexts;
00287 
00288 /// <summary>
00289 /// The <see cref="RenderingThreads"/> that are in charge of the actual rendering.
00290 /// </summary>
00291         private readonly RenderingThread[] RenderingThreads;
00292 
00293 /// <summary>
00294 /// The bounds of the page being rendered.
00295 /// </summary>
00296         private readonly Rectangle PageBounds;
00297 
00298 /// <summary>
```

```
00299 /// Whether the rendered images should be clipped to the bounds of the page being rendered.
00300 /// </summary>
00301         private readonly bool ClipToPageBounds;
00302 
00303 /// <summary>
00304 /// The number of threads that are used to render the image.
00305 /// </summary>
00306         public int ThreadCount { get; }
00307 
00308 /// <summary>
00309 /// If the document is an image, the horizontal resolution of the image.  Otherwise, 72.
00310 /// </summary>
00311         internal double ImageXRes = double.NaN;
00312 
00313 /// <summary>
00314 /// If the document is an image, the vertical resolution of the image.  Otherwise, 72.
00315 /// </summary>
00316         internal double ImageYRes = double.NaN;
00317 
00318 /// <summary>
00319 /// Create a new <see cref="MuPDFMultiThreadedPageRenderer"/> from a specified display list using the
      specified number of threads.
00320 /// </summary>
00321 /// <param name="context">The context that owns the document from which the display list was
      extracted.</param>
00322 /// <param name="displayList">The display list to render.</param>
00323 /// <param name="threadCount">The number of threads to use in the rendering.  This must be
      factorisable using only powers of 2, 3, 5 or 7.  Otherwise, the biggest number smaller than <paramref
      name="threadCount"/> that satisfies this condition is used.</param>
00324 /// <param name="pageBounds">The bounds of the page being rendererd.</param>
00325 /// <param name="clipToPageBounds">A boolean value indicating whether the rendered image should be
      clipped to the original page's bounds.  This can be relevant if the page has been "cropped" by
      altering its mediabox, but otherwise leaving the contents untouched.</param>
00326 /// <param name="imageXRes">If the document is an image, the horizontal resolution of the image.
      Otherwise, 72.</param>
00327 /// <param name="imageYRes">If the document is an image, the vertical resolution of the image.
      Otherwise, 72.</param>
00328         internal MuPDFMultiThreadedPageRenderer(MuPDFContext context, MuPDFDisplayList displayList,
      int threadCount, Rectangle pageBounds, bool clipToPageBounds, double imageXRes, double imageYRes)
00329         {
00330             threadCount = Utils.GetAcceptableNumber(threadCount);
00331 
00332             this.ThreadCount = threadCount;
00333             this.DisplayList = displayList;
00334             this.PageBounds = pageBounds;
00335             this.ClipToPageBounds = clipToPageBounds;
00336 
00337             this.ImageXRes = imageXRes;
00338             this.ImageYRes = imageYRes;
00339 
00340             IntPtr[] contexts = new IntPtr[threadCount];
00341             GCHandle contextsHandle = GCHandle.Alloc(contexts, GCHandleType.Pinned);
00342 
00343             try
00344             {
00345                 ExitCodes result = (ExitCodes)NativeMethods.CloneContext(context.NativeContext,
      threadCount, contextsHandle.AddrOfPinnedObject());
00346 
00347                 switch (result)
00348                 {
00349                     case ExitCodes.EXIT_SUCCESS:
00350                         break;
00351                     case ExitCodes.ERR_CANNOT_INIT_MUTEX:
00352                         throw new MuPDFException("Cannot initalize mutex objects", result);
00353                     case ExitCodes.ERR_CANNOT_CREATE_CONTEXT:
00354                         throw new MuPDFException("Cannot create master context", result);
00355                     case ExitCodes.ERR_CANNOT_CLONE_CONTEXT:
00356                         throw new MuPDFException("Cannot create context clones", result);
00357                     default:
00358                         throw new MuPDFException("Unknown error", result);
00359                 }
00360 
00361                 Contexts = new MuPDFContext[threadCount];
00362                 RenderingThreads = new RenderingThread[threadCount];
00363                 for (int i = 0; i < threadCount; i++)
00364                 {
00365                     Contexts[i] = new MuPDFContext(contexts[i]);
00366                     RenderingThreads[i] = new RenderingThread();
00367                 }
00368             }
00369             finally
00370             {
00371                 contextsHandle.Free();
00372             }
00373         }
00374 
00375 /// <summary>
```

```
00376 /// Render the specified region to an image of the specified size, split in a number of tiles equal to
      the number of threads used by this <see cref="MuPDFMultiThreadedPageRenderer"/>, without marshaling.
      This method will not return until all the rendering threads have finished.
00377 /// </summary>
00378 /// <param name="targetSize">The total size of the image that should be rendered.</param>
00379 /// <param name="region">The region in page units that should be rendered.</param>
00380 /// <param name="destinations">An array containing the addresses of the buffers where the rendered
      tiles will be written.  There must be enough space available in each buffer to write the values for
      all the pixels of the tile, otherwise this will fail catastrophically!
00381 /// As long as the <paramref name="targetSize"/> is the same, the size in pixel of the tiles is
      guaranteed to also be the same.</param>
00382 /// <param name="pixelFormat">The format of the pixel data.</param>
00383         public void Render(RoundedSize targetSize, Rectangle region, IntPtr[] destinations,
      PixelFormats pixelFormat)
00384         {
00385             if (destinations.Length != Contexts.Length)
00386             {
00387                 throw new ArgumentOutOfRangeException(nameof(destinations), destinations.Length, "The
      number of destinations must be equal to the number of rendering threads!");
00388             }
00389 
00390             RoundedRectangle[] targets = targetSize.Split(destinations.Length);
00391 
00392             float zoomX = targetSize.Width / region.Width;
00393             float zoomY = targetSize.Height / region.Height;
00394 
00395             float zoom = (float)Math.Sqrt(zoomX * zoomY);
00396 
00397             Rectangle actualPageArea = new Rectangle(region.X0, region.Y0, region.X0 +
      targetSize.Width / zoom, region.Y0 + targetSize.Height / zoom);
00398 
00399             Rectangle[] origins = actualPageArea.Split(destinations.Length);
00400 
00401             //Make sure that each tile has the expected size in pixel, rounding errors
      notwithstanding.
00402             for (int i = 0; i < origins.Length; i++)
00403             {
00404                 int countBlanks = 0;
00405                 RoundedRectangle roundedOrigin = origins[i].Round(zoom);
00406                 while (roundedOrigin.Width != targets[i].Width || roundedOrigin.Height !=
      targets[i].Height)
00407                 {
00408                     RoundedRectangle oldRoundedOrigin = roundedOrigin;
00409 
00410                     if (roundedOrigin.Width > targets[i].Width)
00411                     {
00412                         if (origins[i].X0 > actualPageArea.X0)
00413                         {
00414                             origins[i] = new Rectangle(origins[i].X0 + 0.5 / zoom, origins[i].Y0,
      origins[i].X1, origins[i].Y1);
00415                         }
00416                         else
00417                         {
00418                             origins[i] = new Rectangle(origins[i].X0, origins[i].Y0, origins[i].X1 -
      0.5 / zoom, origins[i].Y1);
00419                         }
00420                     }
00421                     else if (roundedOrigin.Width < targets[i].Width)
00422                     {
00423                         if (origins[i].X1 < actualPageArea.X1)
00424                         {
00425                             origins[i] = new Rectangle(origins[i].X0, origins[i].Y0, origins[i].X1 +
      0.5 / zoom, origins[i].Y1);
00426                         }
00427                         else
00428                         {
00429                             origins[i] = new Rectangle(origins[i].X0 - 0.5 / zoom, origins[i].Y0,
      origins[i].X1, origins[i].Y1);
00430                         }
00431                     }
00432 
00433 
00434                     if (roundedOrigin.Height > targets[i].Height)
00435                     {
00436                         if (origins[i].Y0 > actualPageArea.Y0)
00437                         {
00438                             origins[i] = new Rectangle(origins[i].X0, origins[i].Y0 + 0.5 / zoom,
      origins[i].X1, origins[i].Y1);
00439                         }
00440                         else
00441                         {
00442                             origins[i] = new Rectangle(origins[i].X0, origins[i].Y0, origins[i].X1,
      origins[i].Y1 - 0.5 / zoom);
00443                         }
00444                     }
00445                     else if (roundedOrigin.Height < targets[i].Height)
00446                     {
```

```
00447                       if (origins[i].X1 < actualPageArea.X1)
00448                       {
00449                           origins[i] = new Rectangle(origins[i].X0, origins[i].Y0, origins[i].X1,
      origins[i].Y1 + 0.5 / zoom);
00450                       }
00451                       else
00452                       {
00453                           origins[i] = new Rectangle(origins[i].X0, origins[i].Y0 - 0.5 / zoom,
      origins[i].X1, origins[i].Y1);
00454                       }
00455                   }
00456
00457                   roundedOrigin = origins[i].Round(zoom);
00458
00459                   if (roundedOrigin.Width == oldRoundedOrigin.Width && roundedOrigin.Height ==
      oldRoundedOrigin.Height && (roundedOrigin.Width != targets[i].Width || roundedOrigin.Height !=
      targets[i].Height))
00460                   {
00461                       countBlanks++;
00462                   }
00463
00464                   if (countBlanks >= 100)
00465                   {
00466                       //It seems that we can't coerce the expected size and the actual size to be
      the same.  Give up.
00467                       return;
00468                   }
00469               }
00470           }
00471
00472           //Start each rendering thread.
00473           for (int i = 0; i < destinations.Length; i++)
00474           {
00475               double dzoom = zoom;
00476               Rectangle origin = origins[i];
00477               if (this.ImageXRes != 72 || this.ImageYRes != 72)
00478               {
00479                   dzoom *= Math.Sqrt(this.ImageXRes * this.ImageYRes) / 72;
00480                   origin = new Rectangle(origin.X0 * 72 / this.ImageXRes, origin.Y0 * 72 /
      this.ImageYRes, origin.X1 * 72 / this.ImageXRes, origin.Y1 * 72 / this.ImageYRes);
00481               }
00482
00483               RenderingThreads[i].Render(Contexts[i].NativeContext, DisplayList, origin,
      (float)dzoom, destinations[i], pixelFormat, this.PageBounds, ClipToPageBounds);
00484           }
00485
00486           //Wait until all the rendering threads have finished.
00487           for (int i = 0; i < destinations.Length; i++)
00488           {
00489               RenderingThreads[i].WaitForRendering();
00490           }
00491       }
00492
00493 /// <summary>
00494 /// Gets an element from a collection of <see cref="System.Span{T}">Span</see>&lt;<see
      cref="byte"/>&gt;
00495 /// </summary>
00496 /// <param name="index">The index of the element to get.</param>
00497 /// <returns>An element from a collection of <see cref="System.Span{T}">Span</see>&lt;<see
      cref="byte"/>&gt;</returns>
00498       public delegate Span<byte> GetSpanItem(int index);
00499
00500 /// <summary>
00501 /// Render the specified region to an image of the specified size, split in a number of tiles equal to
      the number of threads used by this <see cref="MuPDFMultiThreadedPageRenderer"/>, without marshaling.
      This method will not return until all the rendering threads have finished.
00502 /// Since creating an array of <see cref="Span{T}"/> is not allowed, this method returns a delegate
      that accepts an integer parameter (representing the index of the span in the "array") and returns the
      <see cref="Span{T}"/> corresponding to that index.
00503 /// </summary>
00504 /// <param name="targetSize">The total size of the image that should be rendered.</param>
00505 /// <param name="region">The region in page units that should be rendered.</param>
00506 /// <param name="disposables">A collection of <see cref="IDisposable"/>s that can be used to free the
      memory where the rendered tiles are stored.  You should keep track of these and dispose them when you
      have finished working with the images.</param>
00507 /// <param name="pixelFormat">The format of the pixel data.</param>
00508 /// <returns>A delegate that accepts an integer parameter (representing the index of the span in the
      "array") and returns the <see cref="Span{T}"/> corresponding to that index.</returns>
00509       public GetSpanItem Render(RoundedSize targetSize, Rectangle region, out IDisposable[]
      disposables, PixelFormats pixelFormat)
00510       {
00511           RoundedRectangle[] targets = targetSize.Split(this.ThreadCount);
00512
00513           IntPtr[] destinations = new IntPtr[targets.Length];
00514           disposables = new IDisposable[targets.Length];
00515
00516           bool hasAlpha = pixelFormat == PixelFormats.RGBA || pixelFormat == PixelFormats.BGRA;
```

```
00517
00518             int[] sizes = new int[destinations.Length];
00519
00520             for (int i = 0; i < targets.Length; i++)
00521             {
00522                 int allocSize = targets[i].Width * targets[i].Height * (hasAlpha ?  4 :  3);
00523
00524                 sizes[i] = allocSize;
00525                 destinations[i] = Marshal.AllocHGlobal(allocSize);
00526                 disposables[i] = new DisposableIntPtr(destinations[i], allocSize);
00527             }
00528
00529             this.Render(targetSize, region, destinations, pixelFormat);
00530
00531             return i =>
00532             {
00533                 unsafe
00534                 {
00535                     return new Span<byte>((void*)destinations[i], sizes[i]);
00536                 }
00537             };
00538         }
00539
00540 /// <summary>
00541 /// Signal to the rendering threads that they should abort rendering as soon as possible.
00542 /// </summary>
00543         public void Abort()
00544         {
00545             for (int i = 0; i < RenderingThreads.Length; i++)
00546             {
00547                 RenderingThreads[i].AbortRendering();
00548             }
00549         }
00550
00551 /// <summary>
00552 /// Get the current rendering progress of all the threads.
00553 /// </summary>
00554 /// <returns>A <see cref="RenderProgress"/> object containing the rendering progress of all the
      threads.</returns>
00555         public RenderProgress GetProgress()
00556         {
00557             return new RenderProgress((from el in RenderingThreads select
      el.GetProgress()).ToArray());
00558         }
00559
00560         private bool disposedValue;
00561
00562 ///<inheritdoc/>
00563         protected virtual void Dispose(bool disposing)
00564         {
00565             if (!disposedValue)
00566             {
00567                 if (disposing)
00568                 {
00569                     Abort();
00570
00571                     if (RenderingThreads != null)
00572                     {
00573                         for (int i = 0; i < RenderingThreads.Length; i++)
00574                         {
00575                             RenderingThreads[i].Dispose();
00576                         }
00577                     }
00578
00579                     if (Contexts != null)
00580                     {
00581                         for (int i = 0; i < Contexts.Length; i++)
00582                         {
00583                             Contexts[i].Dispose();
00584                         }
00585                     }
00586                 }
00587                 disposedValue = true;
00588             }
00589         }
00590
00591 ///<inheritdoc/>
00592         public void Dispose()
00593         {
00594             Dispose(disposing:  true);
00595             GC.SuppressFinalize(this);
00596         }
00597     }
00598 }
```

8.9 MuPDFPage.cs

```
00001 /*
00002 MuPDFCore - A set of multiplatform .NET Core bindings for MuPDF.
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 System;
00019 using System.Collections;
00020 using System.Collections.Generic;
00021
00022 namespace MuPDFCore
00023 {
00024 /// <summary>
00025 /// A wrapper over a MuPDF page object, which contains information about the page's boundaries.
00026 /// </summary>
00027     public class MuPDFPage :  IDisposable
00028     {
00029 /// <summary>
00030 /// The page's bounds at 72 DPI. Read-only.
00031 /// </summary>
00032         public Rectangle Bounds { get; }
00033
00034 /// <summary>
00035 /// The number of this page in the original document.
00036 /// </summary>
00037         public int PageNumber { get; }
00038
00039 /// <summary>
00040 /// A pointer to the native page object.
00041 /// </summary>
00042         internal readonly IntPtr NativePage;
00043
00044 /// <summary>
00045 /// The context that owns the document from which this page was extracted.
00046 /// </summary>
00047         private readonly MuPDFContext OwnerContext;
00048
00049 /// <summary>
00050 /// The document from which the page was extracted.
00051 /// </summary>
00052         internal readonly MuPDFDocument OwnerDocument;
00053
00054 /// <summary>
00055 /// The page's original bounds.  Read-only.
00056 /// </summary>
00057         internal Rectangle OriginalBounds { get; }
00058
00059 /// <summary>
00060 /// Create a new <see cref="MuPDFPage"/> object from the specified document.
00061 /// </summary>
00062 /// <param name="context">The context that owns the document.</param>
00063 /// <param name="document">The document from which the page should be extracted.</param>
00064 /// <param name="number">The number of the page that should be extracted (starting at 0).</param>
00065         internal MuPDFPage(MuPDFContext context, MuPDFDocument document, int number)
00066         {
00067             if (document.EncryptionState == EncryptionState.Encrypted)
00068             {
00069                 throw new DocumentLockedException("A password is necessary to render the document!");
00070             }
00071
00072             this.OwnerContext = context;
00073             this.OwnerDocument = document;
00074             this.PageNumber = number;
00075
00076             float x = 0;
00077             float y = 0;
00078             float w = 0;
00079             float h = 0;
00080
00081             ExitCodes result = (ExitCodes)NativeMethods.LoadPage(context.NativeContext,
      document.NativeDocument, number, ref NativePage, ref x, ref y, ref w, ref h);
00082
00083             this.Bounds = new Rectangle(Math.Round(x * document.ImageXRes / 72.0 * 1000) / 1000,
      Math.Round(y * document.ImageYRes / 72.0 * 1000) / 1000, Math.Round(w * document.ImageXRes / 72.0 *
```

```
      1000) / 1000, Math.Round(h * document.ImageYRes / 72.0 * 1000) / 1000);
00084             this.OriginalBounds = new Rectangle(x, y, w, h);
00085
00086             switch (result)
00087             {
00088                 case ExitCodes.EXIT_SUCCESS:
00089                     break;
00090                 case ExitCodes.ERR_CANNOT_LOAD_PAGE:
00091                     throw new MuPDFException("Cannot load page", result);
00092                 case ExitCodes.ERR_CANNOT_COMPUTE_BOUNDS:
00093                     throw new MuPDFException("Cannot compute bounds", result);
00094                 default:
00095                     throw new MuPDFException("Unknown error", result);
00096             }
00097         }
00098
00099         private bool disposedValue;
00100
00101 ///<inheritdoc/>
00102         protected virtual void Dispose(bool disposing)
00103         {
00104             if (!disposedValue)
00105             {
00106                 NativeMethods.DisposePage(OwnerContext.NativeContext, NativePage);
00107                 disposedValue = true;
00108             }
00109         }
00110
00111 ///<inheritdoc/>
00112         ~MuPDFPage()
00113         {
00114             if (NativePage != IntPtr.Zero)
00115             {
00116                 Dispose(disposing:  false);
00117             }
00118         }
00119
00120 ///<inheritdoc/>
00121         public void Dispose()
00122         {
00123             Dispose(disposing:  true);
00124             GC.SuppressFinalize(this);
00125         }
00126     }
00127
00128 /// <summary>
00129 /// A lazy collection of <see cref="MuPDFPage"/>s.  Each page is loaded from the document as it is
      requested for the first time.
00130 /// </summary>
00131     public class MuPDFPageCollection :  IReadOnlyList<MuPDFPage>, IDisposable
00132     {
00133 /// <summary>
00134 /// The internal store of the pages.
00135 /// </summary>
00136         private readonly MuPDFPage[] Pages;
00137
00138 /// <summary>
00139 /// The context that owns the document from which the pages were extracted.
00140 /// </summary>
00141         private readonly MuPDFContext OwnerContext;
00142
00143 /// <summary>
00144 /// The document from which the pages were extracted.
00145 /// </summary>
00146         private readonly MuPDFDocument OwnerDocument;
00147
00148 /// <summary>
00149 /// The number of pages in the collection.
00150 /// </summary>
00151         public int Length { get { return Pages.Length; } }
00152
00153 /// <summary>
00154 /// The number of pages in the collection.
00155 /// </summary>
00156         public int Count { get { return Pages.Length; } }
00157
00158 /// <summary>
00159 /// Get a page from the collection.
00160 /// </summary>
00161 /// <param name="index">The number of the page (starting at 0).</param>
00162 /// <returns>The specified <see cref="MuPDFPage"/>.</returns>
00163         public MuPDFPage this[int index]
00164         {
00165             get
00166             {
00167                 if (index < 0 || index > Pages.Length)
00168                 {
```

```
00169                     throw new IndexOutOfRangeException();
00170                 }
00171
00172                 if (Pages[index] == null)
00173                 {
00174                     Pages[index] = new MuPDFPage(OwnerContext, OwnerDocument, index);
00175                 }
00176
00177                 return Pages[index];
00178             }
00179         }
00180
00181 /// <summary>
00182 /// Create a new <see cref="MuPDFPageCollection"/> from the specified document, containing the
      specified number of pages.
00183 /// </summary>
00184 /// <param name="context">The context that owns the document.</param>
00185 /// <param name="document">The document from which the pages should be extracted.</param>
00186 /// <param name="length">The number of pages in the document.</param>
00187         internal MuPDFPageCollection(MuPDFContext context, MuPDFDocument document, int length)
00188         {
00189             Pages = new MuPDFPage[length];
00190             OwnerContext = context;
00191             OwnerDocument = document;
00192         }
00193
00194 ///<inheritdoc/>
00195         public IEnumerator<MuPDFPage> GetEnumerator()
00196         {
00197             for (int i = 0; i < Pages.Length; i++)
00198             {
00199                 if (Pages[i] == null)
00200                 {
00201                     Pages[i] = new MuPDFPage(OwnerContext, OwnerDocument, i);
00202                 }
00203             }
00204
00205             return ((IEnumerable<MuPDFPage>)Pages).GetEnumerator();
00206         }
00207
00208 ///<inheritdoc/>
00209         IEnumerator IEnumerable.GetEnumerator()
00210         {
00211             return GetEnumerator();
00212         }
00213
00214         private bool disposedValue;
00215
00216 ///<inheritdoc/>
00217         protected virtual void Dispose(bool disposing)
00218         {
00219             if (!disposedValue)
00220             {
00221                 if (disposing)
00222                 {
00223                     for (int i = 0; i < Pages.Length; i++)
00224                     {
00225                         Pages[i]?.Dispose();
00226                     }
00227                 }
00228
00229                 disposedValue = true;
00230             }
00231         }
00232
00233 ///<inheritdoc/>
00234         public void Dispose()
00235         {
00236             Dispose(disposing:  true);
00237             GC.SuppressFinalize(this);
00238         }
00239     }
00240 }
```

8.10 MuPDFStructuredTextPage.cs

```
00001 using System;
00002 using System.Collections;
00003 using System.Collections.Generic;
00004 using System.Runtime.InteropServices;
00005 using System.Text;
00006 using System.Text.RegularExpressions;
00007 using System.Threading;
```

```
00008 using System.Threading.Tasks;
00009
00010 namespace MuPDFCore
00011 {
00012 /// <summary>
00013 /// Describes OCR progress.
00014 /// </summary>
00015     public class OCRProgressInfo
00016     {
00017 /// <summary>
00018 /// A value between 0 and 1, indicating how much progress has been completed.
00019 /// </summary>
00020         public double Progress { get; }
00021
00022         internal OCRProgressInfo(double progress)
00023         {
00024             this.Progress = progress;
00025         }
00026     }
00027
00028 /// <summary>
00029 /// Represents a structured representation of the text contained in a page.
00030 /// </summary>
00031     public class MuPDFStructuredTextPage :  IReadOnlyList<MuPDFStructuredTextBlock>
00032     {
00033 /// <summary>
00034 /// The blocks contained in the page.
00035 /// </summary>
00036         public MuPDFStructuredTextBlock[] StructuredTextBlocks { get; private set; }
00037
00038 /// <summary>
00039 /// The number of blocks in the page.
00040 /// </summary>
00041         public int Count =>
      ((IReadOnlyCollection<MuPDFStructuredTextBlock>)StructuredTextBlocks).Count;
00042
00043 /// <summary>
00044 /// Gets the specified block in the page.
00045 /// </summary>
00046 /// <param name="index">The index of the block.</param>
00047 /// <returns>The block with the specified <paramref name="index"/>.</returns>
00048         public MuPDFStructuredTextBlock this[int index] =>
      ((IReadOnlyList<MuPDFStructuredTextBlock>)StructuredTextBlocks)[index];
00049
00050 /// <summary>
00051 /// Gets the specified character in the page.
00052 /// </summary>
00053 /// <param name="address">The address (block, line and character index) of the character.</param>
00054 /// <returns>A <see cref="MuPDFStructuredTextCharacter"/> representing the specified
      character.</returns>
00055         public MuPDFStructuredTextCharacter this[MuPDFStructuredTextAddress address]
00056         {
00057             get
00058             {
00059                 return
      StructuredTextBlocks[address.BlockIndex][address.LineIndex][address.CharacterIndex];
00060             }
00061         }
00062
00063         internal MuPDFStructuredTextPage(MuPDFContext context, MuPDFDisplayList list,
      TesseractLanguage ocrLanguage, double zoom, Rectangle pageBounds, CancellationToken cancellationToken
      = default, IProgress<OCRProgressInfo> progress = null)
00064         {
00065             if (ocrLanguage != null && RuntimeInformation.IsOSPlatform(OSPlatform.Windows) &&
      RuntimeInformation.ProcessArchitecture == Architecture.X86 && (cancellationToken != default ||
      progress != null))
00066             {
00067                 throw new PlatformNotSupportedException("A cancellationToken or a progress callback
      are not supported on Windows x86!");
00068             }
00069
00070             int blockCount = -1;
00071
00072             IntPtr nativeStructuredPage = IntPtr.Zero;
00073
00074             ExitCodes result;
00075
00076             if (ocrLanguage != null)
00077             {
00078                 result = (ExitCodes)NativeMethods.GetStructuredTextPageWithOCR(context.NativeContext,
      list.NativeDisplayList, ref nativeStructuredPage, ref blockCount, (float)zoom, pageBounds.X0,
      pageBounds.Y0, pageBounds.X1, pageBounds.Y1, "TESSDATA_PREFIX=" + ocrLanguage.Prefix,
      ocrLanguage.Language, prog =>
00079                 {
00080                     progress?.Report(new OCRProgressInfo(prog / 100.0));
00081
00082                     if (cancellationToken.IsCancellationRequested)
```

```
00083                     {
00084                         return 1;
00085                     }
00086                     else
00087                     {
00088                         return 0;
00089                     }
00090                 });
00091             }
00092             else
00093             {
00094                 result = (ExitCodes)NativeMethods.GetStructuredTextPage(context.NativeContext,
      list.NativeDisplayList, ref nativeStructuredPage, ref blockCount);
00095             }
00096
00097             cancellationToken.ThrowIfCancellationRequested();
00098
00099             switch (result)
00100             {
00101                 case ExitCodes.EXIT_SUCCESS:
00102                     break;
00103                 case ExitCodes.ERR_CANNOT_CREATE_PAGE:
00104                     throw new MuPDFException("Cannot create page", result);
00105                 case ExitCodes.ERR_CANNOT_POPULATE_PAGE:
00106                     throw new MuPDFException("Cannot populate page", result);
00107                 default:
00108                     throw new MuPDFException("Unknown error", result);
00109             }
00110
00111             IntPtr[] blockPointers = new IntPtr[blockCount];
00112             GCHandle blocksHandle = GCHandle.Alloc(blockPointers, GCHandleType.Pinned);
00113
00114             try
00115             {
00116                 result = (ExitCodes)NativeMethods.GetStructuredTextBlocks(nativeStructuredPage,
      blocksHandle.AddrOfPinnedObject());
00117
00118                 switch (result)
00119                 {
00120                     case ExitCodes.EXIT_SUCCESS:
00121                         break;
00122                     default:
00123                         throw new MuPDFException("Unknown error", result);
00124                 }
00125
00126                 StructuredTextBlocks = new MuPDFStructuredTextBlock[blockCount];
00127
00128                 for (int i = 0; i < blockCount; i++)
00129                 {
00130                     int type = -1;
00131                     float x0 = -1;
00132                     float y0 = -1;
00133                     float x1 = -1;
00134                     float y1 = -1;
00135                     int lineCount = -1;
00136
00137                     result = (ExitCodes)NativeMethods.GetStructuredTextBlock(blockPointers[i], ref
      type, ref x0, ref y0, ref x1, ref y1, ref lineCount);
00138
00139                     switch (result)
00140                     {
00141                         case ExitCodes.EXIT_SUCCESS:
00142                             break;
00143                         default:
00144                             throw new MuPDFException("Unknown error", result);
00145                     }
00146
00147                     Rectangle bBox = new Rectangle(x0, y0, x1, y1);
00148
00149                     switch ((MuPDFStructuredTextBlock.Types)type)
00150                     {
00151                         case MuPDFStructuredTextBlock.Types.Image:
00152                             this.StructuredTextBlocks[i] = new MuPDFImageStructuredTextBlock(bBox);
00153                             break;
00154                         case MuPDFStructuredTextBlock.Types.Text:
00155                             this.StructuredTextBlocks[i] = new MuPDFTextStructuredTextBlock(bBox,
      blockPointers[i], lineCount);
00156                             break;
00157                     }
00158                 }
00159             }
00160             finally
00161             {
00162                 blocksHandle.Free();
00163             }
00164
00165             NativeMethods.DisposeStructuredTextPage(context.NativeContext, nativeStructuredPage);
```

```
00166         }
00167
00168 /// <summary>
00169 /// Gets the address of the character that contains the specified <paramref name="point"/> in page
      units.
00170 /// </summary>
00171 /// <param name="point">The point that must be contained by the character.  This is expressed in page
      units (i.e.  with a zoom factor of 1).</param>
00172 /// <param name="includeImages">If this is <see langword="true"/>, blocks containing images may be
      returned.  Otherwise, only blocks containing text are considered.</param>
00173 /// <returns>The address of the character containing the specified <paramref name="point"/>, or <see
      langword="null"/> if no character contains the <paramref name="point"/>.</returns>
00174         public MuPDFStructuredTextAddress?  GetHitAddress(PointF point, bool includeImages)
00175         {
00176             for (int i = 0; i < this.Count; i++)
00177             {
00178                 if (includeImages || this[i].Type == MuPDFStructuredTextBlock.Types.Text)
00179                 {
00180                     if (this[i].BoundingBox.Contains(point))
00181                     {
00182                         for (int j = 0; j < this[i].Count; j++)
00183                         {
00184                             if (this[i][j].BoundingBox.Contains(point))
00185                             {
00186                                 for (int k = 0; k < this[i][j].Count; k++)
00187                                 {
00188                                     if (this[i][j][k].BoundingQuad.Contains(point))
00189                                     {
00190                                         return new MuPDFStructuredTextAddress(i, j, k);
00191                                     }
00192                                 }
00193                             }
00194                         }
00195                     }
00196                 }
00197             }
00198
00199             return null;
00200         }
00201
00202 /// <summary>
00203 /// Gets the address of the character that contains the specified <paramref name="point"/> in page
      units.
00204 /// </summary>
00205 /// <param name="point">The point that must be closest to the character.  This is expressed in page
      units (i.e.  with a zoom factor of 1).</param>
00206 /// <param name="includeImages">If this is <see langword="true"/>, blocks containing images may be
      returned.  Otherwise, only blocks containing text are considered.</param>
00207 /// <returns>The address of the character closest to the specified <paramref name="point"/> This is
      <see langword="null"/> only if the page contains no characters.</returns>
00208         public MuPDFStructuredTextAddress?  GetClosestHitAddress(PointF point, bool includeImages)
00209         {
00210             float minDistance = float.MaxValue;
00211             MuPDFStructuredTextAddress?  closestHit = null;
00212
00213             float minBlockDistance = float.MaxValue;
00214             float minLineDistance = float.MaxValue;
00215
00216             for (int i = 0; i < this.Count; i++)
00217             {
00218                 if (includeImages || this[i].Type == MuPDFStructuredTextBlock.Types.Text)
00219                 {
00220                     float dx = Math.Max(0, Math.Max(this[i].BoundingBox.X0 - point.X, point.X -
      this[i].BoundingBox.X1));
00221                     float dy = Math.Max(0, Math.Max(this[i].BoundingBox.Y0 - point.Y, point.Y -
      this[i].BoundingBox.Y1));
00222                     float blockDist = dx * dx + dy * dy;
00223
00224                     if (this[i].BoundingBox.Contains(point) || blockDist < minBlockDistance)
00225                     {
00226                         if (blockDist < minBlockDistance)
00227                         {
00228                             minBlockDistance = blockDist;
00229                             minLineDistance = float.MaxValue;
00230                         }
00231
00232                         for (int j = 0; j < this[i].Count; j++)
00233                         {
00234                             dx = Math.Max(0, Math.Max(this[i][j].BoundingBox.X0 - point.X, point.X -
      this[i][j].BoundingBox.X1));
00235                             dy = Math.Max(0, Math.Max(this[i][j].BoundingBox.Y0 - point.Y, point.Y -
      this[i][j].BoundingBox.Y1));
00236                             float lineDist = dx * dx + dy * dy;
00237
00238                             if (this[i][j].BoundingBox.Contains(point) || lineDist < minLineDistance)
00239                             {
00240                                 if (lineDist < minLineDistance)
```

```
00241                                 {
00242                                     minLineDistance = lineDist;
00243                                 }
00244 
00245                                 for (int k = 0; k < this[i][j].Count; k++)
00246                                 {
00247                                     if (this[i][j][k].BoundingQuad.Contains(point))
00248                                     {
00249                                         return new MuPDFStructuredTextAddress(i, j, k);
00250                                     }
00251                                     else
00252                                     {
00253                                         //The quads should be small enough that the error due to only checking vertices and not sides is negligible.  Also, since the square root is monotonous, we can skip it.
00254                                         float minDist = (point.X - this[i][j][k].BoundingQuad.UpperLeft.X) * (point.X - this[i][j][k].BoundingQuad.UpperLeft.X) + (point.Y - this[i][j][k].BoundingQuad.UpperLeft.Y) * (point.Y - this[i][j][k].BoundingQuad.UpperLeft.Y);
00255                                         minDist = Math.Min(minDist, (point.X - this[i][j][k].BoundingQuad.UpperRight.X) * (point.X - this[i][j][k].BoundingQuad.UpperRight.X) + (point.Y - this[i][j][k].BoundingQuad.UpperRight.Y) * (point.Y - this[i][j][k].BoundingQuad.UpperRight.Y));
00256                                         minDist = Math.Min(minDist, (point.X - this[i][j][k].BoundingQuad.LowerRight.X) * (point.X - this[i][j][k].BoundingQuad.LowerRight.X) + (point.Y - this[i][j][k].BoundingQuad.LowerRight.Y) * (point.Y - this[i][j][k].BoundingQuad.LowerRight.Y));
00257                                         minDist = Math.Min(minDist, (point.X - this[i][j][k].BoundingQuad.LowerLeft.X) * (point.X - this[i][j][k].BoundingQuad.LowerLeft.X) + (point.Y - this[i][j][k].BoundingQuad.LowerLeft.Y) * (point.Y - this[i][j][k].BoundingQuad.LowerLeft.Y));
00258 
00259                                         if (minDist < minDistance)
00260                                         {
00261                                             minDistance = minDist;
00262                                             closestHit = new MuPDFStructuredTextAddress(i, j, k);
00263                                         }
00264                                     }
00265                                 }
00266                             }
00267                         }
00268                     }
00269                 }
00270             }
00271 
00272             return closestHit;
00273         }
00274 
00275 /// <summary>
00276 /// Gets a collection of <see cref="Quad"/>s delimiting the specified character <paramref name="range"/>.  Where possible, these are collapsed at the line and block level.  Each <see cref="Quad"/> may or may not be a rectangle.
00277 /// </summary>
00278 /// <param name="range">A <see cref="MuPDFStructuredTextAddressSpan"/> representing the character range</param>
00279 /// <param name="includeImages">If this is <see langword="true"/>, the bounding boxes for blocks containing images are also returned.  Otherwise, only blocks containing text are considered.</param>
00280 /// <returns>A lazy collection of <see cref="Quad"/>s delimiting the characters in the specified <paramref name="includeImages"/>.</returns>
00281         public IEnumerable<Quad> GetHighlightQuads(MuPDFStructuredTextAddressSpan range, bool includeImages)
00282         {
00283             if (range == null || range.End == null)
00284             {
00285                 yield break;
00286             }
00287 
00288             MuPDFStructuredTextAddress rangeStart = range.Start;
00289             MuPDFStructuredTextAddress rangeEnd = range.End.Value;
00290 
00291             if (rangeEnd < rangeStart)
00292             {
00293                 MuPDFStructuredTextAddress temp = rangeStart;
00294                 rangeStart = rangeEnd;
00295                 rangeEnd = temp;
00296             }
00297 
00298             if (rangeStart.BlockIndex != rangeEnd.BlockIndex)
00299             {
00300                 //Add remaining part of this block
00301                 if (rangeStart.LineIndex == 0 && rangeStart.CharacterIndex == 0)
00302                 {
00303                     if (includeImages || this[rangeStart.BlockIndex].Type == MuPDFStructuredTextBlock.Types.Text)
00304                     {
00305                         yield return this[rangeStart.BlockIndex].BoundingBox.ToQuad();
00306                     }
```

```
00307                 }
00308                 else
00309                 {
00310                     if (rangeStart.CharacterIndex == 0)
00311                     {
00312                         yield return
      this[rangeStart.BlockIndex][rangeStart.LineIndex].BoundingBox.ToQuad();
00313                     }
00314                     else
00315                     {
00316                         for (int i = rangeStart.CharacterIndex; i <
      this[rangeStart.BlockIndex][rangeStart.LineIndex].Count; i++)
00317                         {
00318                             yield return
      this[rangeStart.BlockIndex][rangeStart.LineIndex][i].BoundingQuad;
00319                         }
00320                     }
00321 
00322                     for (int j = rangeStart.LineIndex + 1; j < this[rangeStart.BlockIndex].Count; j++)
00323                     {
00324                         yield return this[rangeStart.BlockIndex][j].BoundingBox.ToQuad();
00325                     }
00326                 }
00327 
00328                 //Add full blocks in the middle
00329                 for (int i = rangeStart.BlockIndex + 1; i < rangeEnd.BlockIndex; i++)
00330                 {
00331                     if (includeImages || this[i].Type == MuPDFStructuredTextBlock.Types.Text)
00332                     {
00333                         yield return this[i].BoundingBox.ToQuad();
00334                     }
00335                 }
00336 
00337                 rangeStart = new MuPDFStructuredTextAddress(rangeEnd.BlockIndex, 0, 0);
00338             }
00339 
00340             if (includeImages || this[rangeStart.BlockIndex].Type ==
      MuPDFStructuredTextBlock.Types.Text)
00341             {
00342                 if (rangeStart.LineIndex != rangeEnd.LineIndex)
00343                 {
00344                     //Add remaining part of this line
00345                     if (rangeStart.CharacterIndex == 0)
00346                     {
00347                         yield return
      this[rangeStart.BlockIndex][rangeStart.LineIndex].BoundingBox.ToQuad();
00348                     }
00349                     else
00350                     {
00351                         for (int i = rangeStart.CharacterIndex; i <
      this[rangeStart.BlockIndex][rangeStart.LineIndex].Count; i++)
00352                         {
00353                             yield return
      this[rangeStart.BlockIndex][rangeStart.LineIndex][i].BoundingQuad;
00354                         }
00355                     }
00356 
00357                     //Add full lines in the middle
00358                     for (int j = rangeStart.LineIndex + 1; j < rangeEnd.LineIndex; j++)
00359                     {
00360                         yield return this[rangeStart.BlockIndex][j].BoundingBox.ToQuad();
00361                     }
00362 
00363                     rangeStart = new MuPDFStructuredTextAddress(rangeEnd.BlockIndex,
      rangeEnd.LineIndex, 0);
00364                 }
00365 
00366                 //Add remaining part of this line
00367                 if (rangeStart.CharacterIndex == 0 && rangeEnd.CharacterIndex ==
      this[rangeStart.BlockIndex][rangeStart.LineIndex].Count - 1)
00368                 {
00369                     yield return
      this[rangeStart.BlockIndex][rangeStart.LineIndex].BoundingBox.ToQuad();
00370                 }
00371                 else
00372                 {
00373                     for (int j = rangeStart.CharacterIndex; j <= rangeEnd.CharacterIndex; j++)
00374                     {
00375                         yield return
      this[rangeStart.BlockIndex][rangeStart.LineIndex][j].BoundingQuad;
00376                     }
00377                 }
00378             }
00379         }
00380 
00381 /// <summary>
00382 /// Gets the text corresponding to the specified character <paramref name="range"/>.  Blocks
```

```
containing images are ignored.
00383 /// </summary>
00384 /// <param name="range">A <see cref="MuPDFStructuredTextAddressSpan"/> representing the range of text
to extract.</param>
00385 /// <returns>A string representation of the text contained in the specified <paramref
name="range"/>.</returns>
00386         public string GetText(MuPDFStructuredTextAddressSpan range)
00387         {
00388             if (range == null || range.End == null)
00389             {
00390                 return null;
00391             }
00392 
00393             MuPDFStructuredTextAddress selectionStart = range.Start;
00394             MuPDFStructuredTextAddress selectionEnd = range.End.Value;
00395 
00396             if (selectionEnd < selectionStart)
00397             {
00398                 MuPDFStructuredTextAddress temp = selectionStart;
00399                 selectionStart = selectionEnd;
00400                 selectionEnd = temp;
00401             }
00402 
00403             StringBuilder builder = new StringBuilder();
00404 
00405             if (selectionStart.BlockIndex != selectionEnd.BlockIndex)
00406             {
00407                 //Add remaining part of this block
00408                 if (selectionStart.LineIndex == 0 && selectionStart.CharacterIndex == 0)
00409                 {
00410                     if (this[selectionStart.BlockIndex].Type == MuPDFStructuredTextBlock.Types.Text)
00411                     {
00412 
builder.Append(((MuPDFTextStructuredTextBlock)this[selectionStart.BlockIndex]).ToString());
00413                     }
00414                 }
00415                 else
00416                 {
00417                     if (selectionStart.CharacterIndex == 0)
00418                     {
00419 
builder.AppendLine(this[selectionStart.BlockIndex][selectionStart.LineIndex].ToString());
00420                     }
00421                     else
00422                     {
00423                         for (int i = selectionStart.CharacterIndex; i <
this[selectionStart.BlockIndex][selectionStart.LineIndex].Count; i++)
00424                         {
00425 
builder.Append(this[selectionStart.BlockIndex][selectionStart.LineIndex][i].ToString());
00426                         }
00427                         builder.AppendLine();
00428                     }
00429 
00430                     for (int j = selectionStart.LineIndex + 1; j <
this[selectionStart.BlockIndex].Count; j++)
00431                     {
00432                         builder.AppendLine(this[selectionStart.BlockIndex][j].ToString());
00433                     }
00434                 }
00435 
00436                 //Add full blocks in the middle
00437                 for (int i = selectionStart.BlockIndex + 1; i < selectionEnd.BlockIndex; i++)
00438                 {
00439                     if (this[i].Type == MuPDFStructuredTextBlock.Types.Text)
00440                     {
00441                         builder.Append(this[i].ToString());
00442                     }
00443                 }
00444 
00445                 selectionStart = new MuPDFStructuredTextAddress(selectionEnd.BlockIndex, 0, 0);
00446             }
00447 
00448             if (this[selectionStart.BlockIndex].Type == MuPDFStructuredTextBlock.Types.Text)
00449             {
00450                 if (selectionStart.LineIndex != selectionEnd.LineIndex)
00451                 {
00452                     //Add remaining part of this line
00453                     if (selectionStart.CharacterIndex == 0)
00454                     {
00455 
builder.AppendLine(this[selectionStart.BlockIndex][selectionStart.LineIndex].ToString());
00456                     }
00457                     else
00458                     {
00459                         for (int i = selectionStart.CharacterIndex; i <
this[selectionStart.BlockIndex][selectionStart.LineIndex].Count; i++)
```

```
00460                         {
00461
      builder.Append(this[selectionStart.BlockIndex][selectionStart.LineIndex][i].ToString());
00462                         }
00463                         builder.AppendLine();
00464                     }
00465
00466                     //Add full lines in the middle
00467                     for (int j = selectionStart.LineIndex + 1; j < selectionEnd.LineIndex; j++)
00468                     {
00469                         builder.AppendLine(this[selectionStart.BlockIndex][j].ToString());
00470                     }
00471
00472                     selectionStart = new MuPDFStructuredTextAddress(selectionEnd.BlockIndex,
      selectionEnd.LineIndex, 0);
00473                 }
00474
00475                 //Add remaining part of this line
00476                 if (selectionStart.CharacterIndex == 0 && selectionEnd.CharacterIndex ==
      this[selectionStart.BlockIndex][selectionStart.LineIndex].Count - 1)
00477                 {
00478
      builder.Append(this[selectionStart.BlockIndex][selectionStart.LineIndex].ToString());
00479                 }
00480                 else
00481                 {
00482                     for (int j = selectionStart.CharacterIndex; j <= selectionEnd.CharacterIndex; j++)
00483                     {
00484
      builder.Append(this[selectionStart.BlockIndex][selectionStart.LineIndex][j].ToString());
00485                     }
00486                 }
00487             }
00488
00489             return builder.ToString();
00490         }
00491
00492 /// <summary>
00493 /// Searches for the specified <see cref="Regex"/> in the text of the page.  A single match cannot
      span multiple lines.
00494 /// </summary>
00495 /// <param name="needle">The <see cref="Regex"/> to search for.</param>
00496 /// <returns>A lazy collection of <see cref="MuPDFStructuredTextAddressSpan"/>s representing all the
      occurrences of the <paramref name="needle"/> in the text.</returns>
00497         public IEnumerable<MuPDFStructuredTextAddressSpan> Search(Regex needle)
00498         {
00499
00500             for (int i = 0; i < this.Count; i++)
00501             {
00502                 if (this[i].Type == MuPDFStructuredTextBlock.Types.Text)
00503                 {
00504                     for (int j = 0; j < this[i].Count; j++)
00505                     {
00506                         foreach (Match match in needle.Matches(this[i][j].Text))
00507                         {
00508                             if (match.Success)
00509                             {
00510                                 int startIndex = 0;
00511                                 int kStart = 0;
00512
00513                                 while (startIndex < match.Index)
00514                                 {
00515                                     startIndex += this[i][j][kStart].Character.Length;
00516                                     kStart++;
00517                                 }
00518
00519                                 if (startIndex > match.Index)
00520                                 {
00521                                     kStart--;
00522                                 }
00523
00524                                 int length = 0;
00525                                 int kEnd = kStart - 1;
00526
00527                                 while (length < match.Length)
00528                                 {
00529                                     kEnd++;
00530                                     length += this[i][j][kEnd].Character.Length;
00531                                 }
00532
00533                                 yield return new MuPDFStructuredTextAddressSpan(new
      MuPDFStructuredTextAddress(i, j, kStart), new MuPDFStructuredTextAddress(i, j, kEnd));
00534                             }
00535                         }
00536                     }
00537                 }
00538             }
```

```
00539         }
00540
00541 /// <inheritdoc/>
00542         public IEnumerator<MuPDFStructuredTextBlock> GetEnumerator()
00543         {
00544             return ((IEnumerable<MuPDFStructuredTextBlock>)StructuredTextBlocks).GetEnumerator();
00545         }
00546
00547         IEnumerator IEnumerable.GetEnumerator()
00548         {
00549             return StructuredTextBlocks.GetEnumerator();
00550         }
00551     }
00552
00553 /// <summary>
00554 /// Represents a structured text block containing text or an image.
00555 /// </summary>
00556     public abstract class MuPDFStructuredTextBlock :  IReadOnlyList<MuPDFStructuredTextLine>
00557     {
00558 /// <summary>
00559 /// Defines the type of the block.
00560 /// </summary>
00561         public enum Types
00562         {
00563 /// <summary>
00564 /// The block contains text.
00565 /// </summary>
00566             Text = 0,
00567
00568 /// <summary>
00569 /// The block contains an image.
00570 /// </summary>
00571             Image = 1
00572         }
00573
00574 /// <summary>
00575 /// The type of the block.
00576 /// </summary>
00577         public abstract Types Type { get; }
00578
00579 /// <summary>
00580 /// The bounding box of the block.
00581 /// </summary>
00582         public Rectangle BoundingBox { get; }
00583
00584 /// <summary>
00585 /// The number of lines in the block.
00586 /// </summary>
00587         public abstract int Count { get; }
00588
00589 /// <summary>
00590 /// Gets the specified line from the block.
00591 /// </summary>
00592 /// <param name="index">The index of the line to extract.</param>
00593 /// <returns>The <see cref="MuPDFStructuredTextLine"/> with the specified <paramref
      name="index"/>.</returns>
00594         public abstract MuPDFStructuredTextLine this[int index] { get; }
00595
00596         internal MuPDFStructuredTextBlock() { }
00597
00598         internal MuPDFStructuredTextBlock(Rectangle boundingBox)
00599         {
00600             this.BoundingBox = boundingBox;
00601         }
00602
00603 /// <inheritdoc/>
00604         public abstract IEnumerator<MuPDFStructuredTextLine> GetEnumerator();
00605
00606         IEnumerator IEnumerable.GetEnumerator()
00607         {
00608             return this.GetEnumerator();
00609         }
00610     }
00611
00612 /// <summary>
00613 /// Represents a block containing a single image.  The block contains a single line with a single
      character.
00614 /// </summary>
00615     public class MuPDFImageStructuredTextBlock :  MuPDFStructuredTextBlock
00616     {
00617 /// <inheritdoc/>
00618         public override Types Type => Types.Image;
00619
00620 /// <inheritdoc/>
00621         public override int Count => 1;
00622
00623         private readonly MuPDFStructuredTextLine Line;
```

```
00624
00625 /// <inheritdoc/>
00626         public override MuPDFStructuredTextLine this[int index]
00627         {
00628             get
00629             {
00630                 if (index == 0)
00631                 {
00632                     return Line;
00633                 }
00634                 else
00635                 {
00636                     throw new IndexOutOfRangeException("A structured text block containing an image
      only has one line!");
00637                 }
00638             }
00639         }
00640
00641         internal MuPDFImageStructuredTextBlock(Rectangle boundingBox) :  base(boundingBox)
00642         {
00643             this.Line = new MuPDFStructuredTextLine(this.BoundingBox);
00644         }
00645
00646 /// <inheritdoc/>
00647         public override IEnumerator<MuPDFStructuredTextLine> GetEnumerator()
00648         {
00649             return ((IEnumerable<MuPDFStructuredTextLine>)new MuPDFStructuredTextLine[] { Line
      }).GetEnumerator();
00650         }
00651     }
00652
00653 /// <summary>
00654 /// Represents a block containing multiple lines of text (typically a paragraph).
00655 /// </summary>
00656     public class MuPDFTextStructuredTextBlock :  MuPDFStructuredTextBlock
00657     {
00658 /// <inheritdoc/>
00659         public override Types Type => Types.Text;
00660
00661 /// <summary>
00662 /// The lines of text in the block.
00663 /// </summary>
00664         public MuPDFStructuredTextLine[] Lines { get; }
00665
00666 /// <inheritdoc/>
00667         public override int Count => ((IReadOnlyCollection<MuPDFStructuredTextLine>)Lines).Count;
00668
00669 /// <inheritdoc/>
00670         public override MuPDFStructuredTextLine this[int index] =>
      ((IReadOnlyList<MuPDFStructuredTextLine>)Lines)[index];
00671
00672         internal MuPDFTextStructuredTextBlock(Rectangle boundingBox, IntPtr blockPointer, int
      lineCount) :  base(boundingBox)
00673         {
00674             IntPtr[] linePointers = new IntPtr[lineCount];
00675             GCHandle linesHandle = GCHandle.Alloc(linePointers, GCHandleType.Pinned);
00676
00677             try
00678             {
00679                 ExitCodes result = (ExitCodes)NativeMethods.GetStructuredTextLines(blockPointer,
      linesHandle.AddrOfPinnedObject());
00680
00681                 switch (result)
00682                 {
00683                     case ExitCodes.EXIT_SUCCESS:
00684                         break;
00685                     default:
00686                         throw new MuPDFException("Unknown error", result);
00687                 }
00688
00689                 Lines = new MuPDFStructuredTextLine[lineCount];
00690
00691                 for (int i = 0; i < lineCount; i++)
00692                 {
00693                     int wmode = -1;
00694                     float x0 = -1;
00695                     float y0 = -1;
00696                     float x1 = -1;
00697                     float y1 = -1;
00698
00699                     float x = -1;
00700                     float y = -1;
00701
00702                     int charCount = -1;
00703
00704                     result = (ExitCodes)NativeMethods.GetStructuredTextLine(linePointers[i], ref
      wmode, ref x0, ref y0, ref x1, ref y1, ref x, ref y, ref charCount);
```

```
00705
00706                     switch (result)
00707                     {
00708                         case ExitCodes.EXIT_SUCCESS:
00709                             break;
00710                         default:
00711                             throw new MuPDFException("Unknown error", result);
00712                     }
00713
00714                     Rectangle bBox = new Rectangle(x0, y0, x1, y1);
00715                     PointF direction = new PointF(x, y);
00716
00717                     Lines[i] = new MuPDFStructuredTextLine(linePointers[i],
      (MuPDFStructuredTextLine.WritingModes)wmode, direction, bBox, charCount);
00718                 }
00719             }
00720             finally
00721             {
00722                 linesHandle.Free();
00723             }
00724         }
00725
00726 /// <inheritdoc/>
00727         public override IEnumerator<MuPDFStructuredTextLine> GetEnumerator()
00728         {
00729             return ((IEnumerable<MuPDFStructuredTextLine>)Lines).GetEnumerator();
00730         }
00731
00732 /// <summary>
00733 /// Returns the text contained in the block as a <see cref="string"/>.
00734 /// </summary>
00735 /// <returns>The text contained in the block as a <see cref="string"/>.  If the block contains at
      least one line, the return value has a line terminator at the end.</returns>
00736         public override string ToString()
00737         {
00738             StringBuilder text = new StringBuilder();
00739
00740             foreach (MuPDFStructuredTextLine line in this)
00741             {
00742                 text.AppendLine(line.Text);
00743             }
00744
00745             return text.ToString();
00746         }
00747     }
00748
00749 /// <summary>
00750 /// Represents a single line of text (i.e.  characters that share a common baseline).
00751 /// </summary>
00752     public class MuPDFStructuredTextLine :  IReadOnlyList<MuPDFStructuredTextCharacter>
00753     {
00754 /// <summary>
00755 /// Defines the writing mode of the text.
00756 /// </summary>
00757         public enum WritingModes
00758         {
00759 /// <summary>
00760 /// The text is written horizontally.
00761 /// </summary>
00762             Horizontal = 0,
00763
00764 /// <summary>
00765 /// The text is written vertically.
00766 /// </summary>
00767             Vertical = 1
00768         }
00769
00770 /// <summary>
00771 /// The writing mode of the text.
00772 /// </summary>
00773         public WritingModes WritingMode { get; }
00774
00775 /// <summary>
00776 /// The normalised direction of the text baseline.
00777 /// </summary>
00778         public PointF Direction { get; }
00779
00780 /// <summary>
00781 /// The bounding box of the line.
00782 /// </summary>
00783         public Rectangle BoundingBox { get; }
00784
00785 /// <summary>
00786 /// The characters contained in the line.
00787 /// </summary>
00788         public MuPDFStructuredTextCharacter[] Characters { get; }
00789
```

```
00790 /// <summary>
00791 /// A string representation of the characters contained in the line.
00792 /// </summary>
00793         public string Text { get; }
00794
00795 /// <summary>
00796 /// The number of characters in the line.
00797 /// </summary>
00798         public int Count => ((IReadOnlyCollection<MuPDFStructuredTextCharacter>)Characters).Count;
00799
00800 /// <summary>
00801 /// Gets the specified character from the line.
00802 /// </summary>
00803 /// <param name="index">The index of the character.</param>
00804 /// <returns>The <see cref="MuPDFStructuredTextCharacter"/> with the specified <paramref
      name="index"/>.</returns>
00805         public MuPDFStructuredTextCharacter this[int index] =>
      ((IReadOnlyList<MuPDFStructuredTextCharacter>)Characters)[index];
00806
00807         internal MuPDFStructuredTextLine(Rectangle boundingBox)
00808         {
00809             this.BoundingBox = boundingBox;
00810             this.Characters = new MuPDFStructuredTextCharacter[]
00811             {
00812                 new MuPDFStructuredTextCharacter(0, -1, new PointF(boundingBox.X0, boundingBox.Y1),
      new Quad(new PointF(boundingBox.X0, boundingBox.Y1), new PointF(boundingBox.X0, boundingBox.Y0), new
      PointF(boundingBox.X1, boundingBox.Y0), new PointF(boundingBox.X1, boundingBox.Y1)), 9)
00813             };
00814         }
00815
00816         internal MuPDFStructuredTextLine(IntPtr linePointer, WritingModes writingMode, PointF
      direction, Rectangle boundingBox, int charCount)
00817         {
00818             this.WritingMode = writingMode;
00819             this.Direction = direction;
00820             this.BoundingBox = boundingBox;
00821
00822             IntPtr[] charPointers = new IntPtr[charCount];
00823             GCHandle charsHandle = GCHandle.Alloc(charPointers, GCHandleType.Pinned);
00824
00825             try
00826             {
00827                 ExitCodes result = (ExitCodes)NativeMethods.GetStructuredTextChars(linePointer,
      charsHandle.AddrOfPinnedObject());
00828
00829                 switch (result)
00830                 {
00831                     case ExitCodes.EXIT_SUCCESS:
00832                         break;
00833                     default:
00834                         throw new MuPDFException("Unknown error", result);
00835                 }
00836
00837                 Characters = new MuPDFStructuredTextCharacter[charCount];
00838
00839                 StringBuilder textBuilder = new StringBuilder(charCount);
00840
00841                 for (int i = 0; i < charCount; i++)
00842                 {
00843                     int c = -1;
00844                     int color = -1;
00845                     float originX = -1;
00846                     float originY = -1;
00847                     float size = -1;
00848                     float llX = -1;
00849                     float llY = -1;
00850                     float ulX = -1;
00851                     float ulY = -1;
00852                     float urX = -1;
00853                     float urY = -1;
00854                     float lrX = -1;
00855                     float lrY = -1;
00856
00857                     result = (ExitCodes)NativeMethods.GetStructuredTextChar(charPointers[i], ref c,
      ref color, ref originX, ref originY, ref size, ref llX, ref llY, ref ulX, ref ulY, ref urX, ref urY,
      ref lrX, ref lrY);
00858
00859                     switch (result)
00860                     {
00861                         case ExitCodes.EXIT_SUCCESS:
00862                             break;
00863                         default:
00864                             throw new MuPDFException("Unknown error", result);
00865                     }
00866
00867                     Quad quad = new Quad(new PointF(llX, llY), new PointF(ulX, ulY), new PointF(urX,
      urY), new PointF(lrX, lrY));
```

```
00868                     PointF origin = new PointF(originX, originY);
00869
00870                     Characters[i] = new MuPDFStructuredTextCharacter(c, color, origin, quad, size);
00871                     textBuilder.Append(Characters[i].Character);
00872                 }
00873
00874                 this.Text = textBuilder.ToString();
00875             }
00876             finally
00877             {
00878                 charsHandle.Free();
00879             }
00880         }
00881
00882 /// <summary>
00883 /// Returns a string representation of the line.
00884 /// </summary>
00885 /// <returns>A string representation of the line.</returns>
00886         public override string ToString()
00887         {
00888             return this.Text;
00889         }
00890
00891 /// <inheritdoc/>
00892         public IEnumerator<MuPDFStructuredTextCharacter> GetEnumerator()
00893         {
00894             return ((IEnumerable<MuPDFStructuredTextCharacter>)Characters).GetEnumerator();
00895         }
00896
00897         IEnumerator IEnumerable.GetEnumerator()
00898         {
00899             return Characters.GetEnumerator();
00900         }
00901     }
00902
00903 /// <summary>
00904 /// Represents a single text character.
00905 /// </summary>
00906     public class MuPDFStructuredTextCharacter
00907     {
00908 /// <summary>
00909 /// The unicode code point of the character.
00910 /// </summary>
00911         public int CodePoint { get; }
00912
00913 /// <summary>
00914 /// A string representation of the character.  It may consist of a single <see cref="char"/> or of a
      surrogate pair of <see cref="char"/>s.
00915 /// </summary>
00916         public string Character { get; }
00917
00918 /// <summary>
00919 /// An sRGB hex representation of the colour of the character.
00920 /// </summary>
00921         public int Color { get; }
00922
00923 /// <summary>
00924 /// The baseline origin of the character.
00925 /// </summary>
00926         public PointF Origin { get; }
00927
00928 /// <summary>
00929 /// A quadrilater bound for the character.  This may or may not be a rectangle.
00930 /// </summary>
00931         public Quad BoundingQuad { get; }
00932
00933 /// <summary>
00934 /// The size in points of the character.
00935 /// </summary>
00936         public float Size { get; }
00937
00938         internal MuPDFStructuredTextCharacter(int codePoint, int color, PointF origin, Quad
      boundingQuad, float size)
00939         {
00940             this.CodePoint = codePoint;
00941             this.Character = Char.ConvertFromUtf32(codePoint);
00942             this.Color = color;
00943             this.Origin = origin;
00944             this.BoundingQuad = boundingQuad;
00945             this.Size = size;
00946         }
00947
00948 /// <summary>
00949 /// Returns a string representation of the character.
00950 /// </summary>
00951 /// <returns>A string representation of the character.</returns>
00952         public override string ToString()
```

```
00953         {
00954             return this.Character;
00955         }
00956     }
00957
00958 /// <summary>
00959 /// Represents the address of a particular character in a <see cref="MuPDFStructuredTextPage"/>, in
      terms of block index, line index and character index.
00960 /// </summary>
00961     public struct MuPDFStructuredTextAddress :  IComparable<MuPDFStructuredTextAddress>,
      IEquatable<MuPDFStructuredTextAddress>
00962     {
00963 /// <summary>
00964 /// The index of the block.
00965 /// </summary>
00966         public readonly int BlockIndex;
00967
00968 /// <summary>
00969 /// The index of the line within the block.
00970 /// </summary>
00971         public readonly int LineIndex;
00972
00973 /// <summary>
00974 /// The index of the character within the line.
00975 /// </summary>
00976         public readonly int CharacterIndex;
00977
00978 /// <summary>
00979 /// Creates a new <see cref="MuPDFStructuredTextAddress"/> from the specified indices.
00980 /// </summary>
00981 /// <param name="blockIndex">The index of the block.</param>
00982 /// <param name="lineIndex">The index of the line within the block.</param>
00983 /// <param name="characterIndex">The index of the character within the line.</param>
00984         public MuPDFStructuredTextAddress(int blockIndex, int lineIndex, int characterIndex)
00985         {
00986             this.BlockIndex = blockIndex;
00987             this.LineIndex = lineIndex;
00988             this.CharacterIndex = characterIndex;
00989         }
00990
00991 /// <summary>
00992 /// Compares this <see cref="MuPDFStructuredTextAddress"/> with another <see
      cref="MuPDFStructuredTextAddress"/>.
00993 /// </summary>
00994 /// <param name="other">The <see cref="MuPDFStructuredTextAddress"/> to compare with the current
      instance.</param>
00995 /// <returns>-1 if the <paramref name="other"/> <see cref="MuPDFStructuredTextAddress"/> comes after
      the current instance, 1 if it comes before, or 0 if they represent the same address.</returns>
00996         public int CompareTo(MuPDFStructuredTextAddress other)
00997         {
00998             if (this < other)
00999             {
01000                 return -1;
01001             }
01002             else if (this > other)
01003             {
01004                 return 1;
01005             }
01006             else
01007             {
01008                 return 0;
01009             }
01010         }
01011
01012 /// <summary>
01013 /// Compares two <see cref="MuPDFStructuredTextAddress"/>.
01014 /// </summary>
01015 /// <param name="first">The first <see cref="MuPDFStructuredTextAddress"/> to compare.</param>
01016 /// <param name="second">The second <see cref="MuPDFStructuredTextAddress"/> to compare.</param>
01017 /// <returns><see langword="true"/> if the <paramref name="first"/> <see
      cref="MuPDFStructuredTextAddress"/> comes after the <paramref name="second"/> one; otherwise, <see
      langword="false"/>.</returns>
01018         public static bool operator >(MuPDFStructuredTextAddress first, MuPDFStructuredTextAddress
      second)
01019         {
01020             if (first.BlockIndex > second.BlockIndex)
01021             {
01022                 return true;
01023             }
01024             else if (first.BlockIndex < second.BlockIndex)
01025             {
01026                 return false;
01027             }
01028             else
01029             {
01030                 if (first.LineIndex > second.LineIndex)
01031                 {
```

```
01032                   return true;
01033               }
01034               else if (first.LineIndex < second.LineIndex)
01035               {
01036                   return false;
01037               }
01038               else
01039               {
01040                   if (first.CharacterIndex > second.CharacterIndex)
01041                   {
01042                       return true;
01043                   }
01044                   else
01045                   {
01046                       return false;
01047                   }
01048               }
01049           }
01050       }
01051
01052 /// <summary>
01053 /// Compares two <see cref="MuPDFStructuredTextAddress"/>.
01054 /// </summary>
01055 /// <param name="first">The first <see cref="MuPDFStructuredTextAddress"/> to compare.</param>
01056 /// <param name="second">The second <see cref="MuPDFStructuredTextAddress"/> to compare.</param>
01057 /// <returns><see langword="true"/> if the <paramref name="first"/> <see
      cref="MuPDFStructuredTextAddress"/> comes after the <paramref name="second"/> one or if they represent
      the same address; otherwise, <see langword="false"/>.</returns>
01058       public static bool operator >=(MuPDFStructuredTextAddress first, MuPDFStructuredTextAddress
      second)
01059       {
01060           if (first.BlockIndex > second.BlockIndex)
01061           {
01062               return true;
01063           }
01064           else if (first.BlockIndex < second.BlockIndex)
01065           {
01066               return false;
01067           }
01068           else
01069           {
01070               if (first.LineIndex > second.LineIndex)
01071               {
01072                   return true;
01073               }
01074               else if (first.LineIndex < second.LineIndex)
01075               {
01076                   return false;
01077               }
01078               else
01079               {
01080                   if (first.CharacterIndex >= second.CharacterIndex)
01081                   {
01082                       return true;
01083                   }
01084                   else
01085                   {
01086                       return false;
01087                   }
01088               }
01089           }
01090       }
01091
01092 /// <summary>
01093 /// Compares two <see cref="MuPDFStructuredTextAddress"/>.
01094 /// </summary>
01095 /// <param name="first">The first <see cref="MuPDFStructuredTextAddress"/> to compare.</param>
01096 /// <param name="second">The second <see cref="MuPDFStructuredTextAddress"/> to compare.</param>
01097 /// <returns><see langword="true"/> if the <paramref name="first"/> <see
      cref="MuPDFStructuredTextAddress"/> comes before the <paramref name="second"/> one; otherwise, <see
      langword="false"/>.</returns>
01098       public static bool operator <(MuPDFStructuredTextAddress first, MuPDFStructuredTextAddress
      second)
01099       {
01100           if (first.BlockIndex > second.BlockIndex)
01101           {
01102               return false;
01103           }
01104           else if (first.BlockIndex < second.BlockIndex)
01105           {
01106               return true;
01107           }
01108           else
01109           {
01110               if (first.LineIndex > second.LineIndex)
01111               {
01112                   return false;
```

```
01113                 }
01114                 else if (first.LineIndex < second.LineIndex)
01115                 {
01116                     return true;
01117                 }
01118                 else
01119                 {
01120                     if (first.CharacterIndex < second.CharacterIndex)
01121                     {
01122                         return true;
01123                     }
01124                     else
01125                     {
01126                         return false;
01127                     }
01128                 }
01129             }
01130         }
01131
01132 /// <summary>
01133 /// Compares two <see cref="MuPDFStructuredTextAddress"/>.
01134 /// </summary>
01135 /// <param name="first">The first <see cref="MuPDFStructuredTextAddress"/> to compare.</param>
01136 /// <param name="second">The second <see cref="MuPDFStructuredTextAddress"/> to compare.</param>
01137 /// <returns><see langword="true"/> if the <paramref name="first"/> <see
      cref="MuPDFStructuredTextAddress"/> comes before the <paramref name="second"/> one or if they
      represent the same address; otherwise, <see langword="false"/>.</returns>
01138         public static bool operator <=(MuPDFStructuredTextAddress first, MuPDFStructuredTextAddress
      second)
01139         {
01140             if (first.BlockIndex > second.BlockIndex)
01141             {
01142                 return false;
01143             }
01144             else if (first.BlockIndex < second.BlockIndex)
01145             {
01146                 return true;
01147             }
01148             else
01149             {
01150                 if (first.LineIndex > second.LineIndex)
01151                 {
01152                     return false;
01153                 }
01154                 else if (first.LineIndex < second.LineIndex)
01155                 {
01156                     return true;
01157                 }
01158                 else
01159                 {
01160                     if (first.CharacterIndex <= second.CharacterIndex)
01161                     {
01162                         return true;
01163                     }
01164                     else
01165                     {
01166                         return false;
01167                     }
01168                 }
01169             }
01170         }
01171
01172 /// <summary>
01173 /// Compares two <see cref="MuPDFStructuredTextAddress"/>.
01174 /// </summary>
01175 /// <param name="first">The first <see cref="MuPDFStructuredTextAddress"/> to compare.</param>
01176 /// <param name="second">The second <see cref="MuPDFStructuredTextAddress"/> to compare.</param>
01177 /// <returns><see langword="true"/> if the two <see cref="MuPDFStructuredTextAddress"/>es represent
      the same address; otherwise, <see langword="false"/>.</returns>
01178         public static bool operator ==(MuPDFStructuredTextAddress first, MuPDFStructuredTextAddress
      second)
01179         {
01180             return first.CharacterIndex == second.CharacterIndex && first.LineIndex ==
      second.LineIndex && first.BlockIndex == second.BlockIndex;
01181         }
01182
01183 /// <summary>
01184 /// Compares two <see cref="MuPDFStructuredTextAddress"/>.
01185 /// </summary>
01186 /// <param name="first">The first <see cref="MuPDFStructuredTextAddress"/> to compare.</param>
01187 /// <param name="second">The second <see cref="MuPDFStructuredTextAddress"/> to compare.</param>
01188 /// <returns><see langword="true"/> if the two <see cref="MuPDFStructuredTextAddress"/>es represent
      different addresses; otherwise, <see langword="false"/>.</returns>
01189         public static bool operator !=(MuPDFStructuredTextAddress first, MuPDFStructuredTextAddress
      second)
01190         {
01191             return first.CharacterIndex != second.CharacterIndex || first.LineIndex !=
```

```
    second.LineIndex || first.BlockIndex != second.BlockIndex;
01192         }
01193
01194 /// <inheritdoc/>
01195         public override int GetHashCode()
01196         {
01197             unchecked
01198             {
01199                 return ((this.BlockIndex * 33 * 33) ^ this.LineIndex * 33) ^ this.CharacterIndex;
01200             }
01201         }
01202
01203 /// <summary>
01204 /// Returns a <see cref="MuPDFStructuredTextAddress"/> corresponding to the next character in the
    specified page.
01205 /// </summary>
01206 /// <param name="page">The page the address refers to.</param>
01207 /// <returns>A <see cref="MuPDFStructuredTextAddress"/> corresponding to the next character in the
    specified page.</returns>
01208         public MuPDFStructuredTextAddress?  Increment(MuPDFStructuredTextPage page)
01209         {
01210             int newBlockIndex = this.BlockIndex;
01211             int newLineIndex = this.LineIndex;
01212             int newCharacterIndex = this.CharacterIndex + 1;
01213
01214             if (page[newBlockIndex][newLineIndex].Count <= newCharacterIndex)
01215             {
01216                 newCharacterIndex = 0;
01217                 newLineIndex++;
01218             }
01219
01220             if (page[newBlockIndex].Count <= newLineIndex)
01221             {
01222                 newLineIndex = 0;
01223                 newBlockIndex++;
01224             }
01225
01226             if (page.Count <= newBlockIndex)
01227             {
01228                 return null;
01229             }
01230
01231             return new MuPDFStructuredTextAddress(newBlockIndex, newLineIndex, newCharacterIndex);
01232         }
01233
01234 /// <summary>
01235 /// Compares the current <see cref="MuPDFStructuredTextAddress"/> with another <see
    cref="MuPDFStructuredTextAddress"/>.
01236 /// </summary>
01237 /// <param name="other">The other <see cref="MuPDFStructuredTextAddress"/> to compare with the current
    instance.</param>
01238 /// <returns><see langword="true"/> if the two <see cref="MuPDFStructuredTextAddress"/>es represent
    the same address; otherwise, <see langword="false"/>.</returns>
01239         public bool Equals(MuPDFStructuredTextAddress other)
01240         {
01241             return this.CharacterIndex == other.CharacterIndex && this.LineIndex == other.LineIndex &&
    this.BlockIndex == other.BlockIndex;
01242         }
01243
01244 /// <inheritdoc/>
01245         public override bool Equals(object other)
01246         {
01247             return other is MuPDFStructuredTextAddress otherAddress && Equals(otherAddress);
01248         }
01249     }
01250
01251 /// <summary>
01252 /// Represents a range of characters in a <see cref="MuPDFStructuredTextPage"/>.
01253 /// </summary>
01254     public class MuPDFStructuredTextAddressSpan
01255     {
01256 /// <summary>
01257 /// The addres of the start of the range.
01258 /// </summary>
01259         public readonly MuPDFStructuredTextAddress Start;
01260
01261 /// <summary>
01262 /// The address of the end of the range (inclusive), or <see langword="null" /> to signify an empty
    range.
01263 /// </summary>
01264         public readonly MuPDFStructuredTextAddress?  End;
01265
01266 /// <summary>
01267 /// Creates a new <see cref="MuPDFStructuredTextAddressSpan"/> corresponding to the specified
    character range.
01268 /// </summary>
01269 /// <param name="start">The addres of the start of the range.</param>
```

```
01270 /// <param name="end">The address of the end of the range (inclusive), or <see langword="null" /> to
      signify an empty range.</param>
01271         public MuPDFStructuredTextAddressSpan(MuPDFStructuredTextAddress start,
      MuPDFStructuredTextAddress?  end)
01272         {
01273             this.Start = start;
01274             this.End = end;
01275         }
01276     }
01277 }
```

8.11 Rectangles.cs

```
00001 /*
00002 MuPDFCore - A set of multiplatform .NET Core bindings for MuPDF.
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 System;
00019
00020 namespace MuPDFCore
00021 {
00022 /// <summary>
00023 /// Represents the size of a rectangle.
00024 /// </summary>
00025     public struct Size
00026     {
00027 /// <summary>
00028 /// The width of the rectangle.
00029 /// </summary>
00030         public float Width;
00031
00032 /// <summary>
00033 /// The height of the rectangle.
00034 /// </summary>
00035         public float Height;
00036
00037 /// <summary>
00038 /// Create a new <see cref="Size"/> with the specified width and height.
00039 /// </summary>
00040 /// <param name="width">The width of the rectangle.</param>
00041 /// <param name="height">The height of the rectangle.</param>
00042         public Size(float width, float height)
00043         {
00044             Width = width;
00045             Height = height;
00046         }
00047
00048 /// <summary>
00049 /// Create a new <see cref="Size"/> with the specified width and height.
00050 /// </summary>
00051 /// <param name="width">The width of the rectangle.</param>
00052 /// <param name="height">The height of the rectangle.</param>
00053         public Size(double width, double height)
00054         {
00055             Width = (float)width;
00056             Height = (float)height;
00057         }
00058
00059 /// <summary>
00060 /// Split the size into the specified number of <see cref="Rectangle"/>s.
00061 /// </summary>
00062 /// <param name="divisions">The number of rectangles in which the size should be split.  This must be
      factorisable using only powers of 2, 3, 5 or 7.  Otherwise, the biggest number smaller than <paramref
      name="divisions"/> that satisfies this condition is used.</param>
00063 /// <returns>An array of <see cref="Rectangle"/>s that when positioned properly cover an area of the
      size of this object.</returns>
00064         public Rectangle[] Split(int divisions)
00065         {
00066             divisions = Utils.GetAcceptableNumber(divisions);
00067
```

```
00068             Rectangle[] tbr = new Rectangle[divisions];
00069
00070             bool isVertical = this.Height > this.Width;
00071
00072             if (divisions == 1)
00073             {
00074                 tbr[0] = new Rectangle(0, 0, Width, Height);
00075             }
00076             else if (divisions == 2)
00077             {
00078                 if (isVertical)
00079                 {
00080                     tbr[0] = new Rectangle(0, 0, Width, Height / 2);
00081                     tbr[1] = new Rectangle(0, Height / 2, Width, Height);
00082                 }
00083                 else
00084                 {
00085                     tbr[0] = new Rectangle(0, 0, Width / 2, Height);
00086                     tbr[1] = new Rectangle(Width / 2, 0, Width, Height);
00087                 }
00088             }
00089             else if (divisions == 3)
00090             {
00091                 if (isVertical)
00092                 {
00093                     tbr[0] = new Rectangle(0, 0, Width / 2, 2 * Height / 3);
00094                     tbr[1] = new Rectangle(Width / 2, 0, Width, 2 * Height / 3);
00095                     tbr[2] = new Rectangle(0, 2 * Height / 3, Width, Height);
00096                 }
00097                 else
00098                 {
00099                     tbr[0] = new Rectangle(0, 0, 2 * Width / 3, Height / 2);
00100                     tbr[1] = new Rectangle(0, Height / 2, 2 * Width / 3, Height);
00101                     tbr[2] = new Rectangle(2 * Width / 3, 0, Width, Height);
00102                 }
00103             }
00104             else if (divisions == 5)
00105             {
00106                 if (isVertical)
00107                 {
00108                     tbr[0] = new Rectangle(0, 0, Width / 2, 2 * Height / 5);
00109                     tbr[1] = new Rectangle(Width / 2, 0, Width, 2 * Height / 5);
00110                     tbr[2] = new Rectangle(0, 2 * Height / 5, Width / 2, 4 * Height / 5);
00111                     tbr[3] = new Rectangle(Width / 2, 2 * Height / 5, Width, 4 * Height / 5);
00112                     tbr[4] = new Rectangle(0, 4 * Height / 5, Width, Height);
00113                 }
00114                 else
00115                 {
00116                     tbr[0] = new Rectangle(0, 0, 2 * Width / 5, Height / 2);
00117                     tbr[1] = new Rectangle(0, Height / 2, 2 * Width / 5, Height);
00118                     tbr[2] = new Rectangle(2 * Width / 5, 0, 4 * Width / 5, Height / 2);
00119                     tbr[3] = new Rectangle(2 * Width / 5, Height / 2, 4 * Width / 5, Height);
00120                     tbr[4] = new Rectangle(4 * Width / 5, 0, Width, Height);
00121                 }
00122             }
00123             else if (divisions == 7)
00124             {
00125                 if (isVertical)
00126                 {
00127                     tbr[0] = new Rectangle(0, 0, Width / 2, 2 * Height / 7);
00128                     tbr[1] = new Rectangle(Width / 2, 0, Width, 2 * Height / 7);
00129                     tbr[2] = new Rectangle(0, 2 * Height / 7, Width / 2, 4 * Height / 7);
00130                     tbr[3] = new Rectangle(Width / 2, 2 * Height / 7, Width, 4 * Height / 7);
00131                     tbr[4] = new Rectangle(0, 4 * Height / 7, Width / 2, 6 * Height / 7);
00132                     tbr[5] = new Rectangle(Width / 2, 4 * Height / 7, Width, 6 * Height / 7);
00133                     tbr[6] = new Rectangle(0, 6 * Height / 7, Width, Height);
00134                 }
00135                 else
00136                 {
00137                     tbr[0] = new Rectangle(0, 0, 2 * Width / 7, Height / 2);
00138                     tbr[1] = new Rectangle(0, Height / 2, 2 * Width / 7, Height);
00139                     tbr[2] = new Rectangle(2 * Width / 7, 0, 4 * Width / 7, Height / 2);
00140                     tbr[3] = new Rectangle(2 * Width / 7, Height / 2, 4 * Width / 7, Height);
00141                     tbr[4] = new Rectangle(4 * Width / 7, 0, 6 * Width / 7, Height / 2);
00142                     tbr[5] = new Rectangle(4 * Width / 7, Height / 2, 6 * Width / 7, Height);
00143                     tbr[6] = new Rectangle(6 * Width / 7, 0, Width, Height);
00144                 }
00145             }
00146             else
00147             {
00148                 for (int divisorInd = 0; divisorInd < Utils.AcceptableDivisors.Length; divisorInd++)
00149                 {
00150                     if (divisions % Utils.AcceptableDivisors[divisorInd] == 0)
00151                     {
00152                         Rectangle[] largerDivisions = this.Split(divisions /
    Utils.AcceptableDivisors[divisorInd]);
00153
```

```
00154                         int pos = 0;
00155
00156                         for (int i = 0; i < largerDivisions.Length; i++)
00157                         {
00158                             Size s = new Size(largerDivisions[i].Width, largerDivisions[i].Height);
00159                             Rectangle[] currDivision = s.Split(Utils.AcceptableDivisors[divisorInd]);
00160
00161                             for (int j = 0; j < currDivision.Length; j++)
00162                             {
00163                                 tbr[pos] = new Rectangle(largerDivisions[i].X0 + currDivision[j].X0,
      largerDivisions[i].Y0 + currDivision[j].Y0, largerDivisions[i].X0 + currDivision[j].X1,
      largerDivisions[i].Y0 + currDivision[j].Y1);
00164                                 pos++;
00165                             }
00166                         }
00167
00168                         break;
00169                     }
00170                 }
00171             }
00172
00173             return tbr;
00174
00175         }
00176     }
00177
00178 /// <summary>
00179 /// Represents the size of a rectangle using only integer numbers.
00180 /// </summary>
00181     public struct RoundedSize
00182     {
00183 /// <summary>
00184 /// The width of the rectangle.
00185 /// </summary>
00186         public int Width;
00187
00188 /// <summary>
00189 /// The height of the rectangle.
00190 /// </summary>
00191         public int Height;
00192
00193 /// <summary>
00194 /// Create a new <see cref="RoundedSize"/> with the specified width and height.
00195 /// </summary>
00196 /// <param name="width">The width of the rectangle.</param>
00197 /// <param name="height">The height of the rectangle.</param>
00198         public RoundedSize(int width, int height)
00199         {
00200             Width = width;
00201             Height = height;
00202         }
00203
00204 /// <summary>
00205 /// Split the size into the specified number of <see cref="RoundedRectangle"/>s.
00206 /// </summary>
00207 /// <param name="divisions">The number of rectangles in which the size should be split.  This must be
      factorisable using only powers of 2, 3, 5 or 7.  Otherwise, the biggest number smaller than <paramref
      name="divisions"/> that satisfies this condition is used.</param>
00208 /// <returns>An array of <see cref="RoundedRectangle"/>s that when positioned properly cover an area
      of the size of this object.</returns>
00209         public RoundedRectangle[] Split(int divisions)
00210         {
00211             divisions = Utils.GetAcceptableNumber(divisions);
00212
00213             RoundedRectangle[] tbr = new RoundedRectangle[divisions];
00214
00215             bool isVertical = this.Height > this.Width;
00216
00217             if (divisions == 1)
00218             {
00219                 tbr[0] = new RoundedRectangle(0, 0, Width, Height);
00220             }
00221             else if (divisions == 2)
00222             {
00223                 if (isVertical)
00224                 {
00225                     tbr[0] = new RoundedRectangle(0, 0, Width, Height / 2);
00226                     tbr[1] = new RoundedRectangle(0, Height / 2, Width, Height);
00227                 }
00228                 else
00229                 {
00230                     tbr[0] = new RoundedRectangle(0, 0, Width / 2, Height);
00231                     tbr[1] = new RoundedRectangle(Width / 2, 0, Width, Height);
00232                 }
00233             }
00234             else if (divisions == 3)
00235             {
```

```
00236                 if (isVertical)
00237                 {
00238                     tbr[0] = new RoundedRectangle(0, 0, Width / 2, 2 * Height / 3);
00239                     tbr[1] = new RoundedRectangle(Width / 2, 0, Width, 2 * Height / 3);
00240                     tbr[2] = new RoundedRectangle(0, 2 * Height / 3, Width, Height);
00241                 }
00242                 else
00243                 {
00244                     tbr[0] = new RoundedRectangle(0, 0, 2 * Width / 3, Height / 2);
00245                     tbr[1] = new RoundedRectangle(0, Height / 2, 2 * Width / 3, Height);
00246                     tbr[2] = new RoundedRectangle(2 * Width / 3, 0, Width, Height);
00247                 }
00248             }
00249             else if (divisions == 5)
00250             {
00251                 if (isVertical)
00252                 {
00253                     tbr[0] = new RoundedRectangle(0, 0, Width / 2, 2 * Height / 5);
00254                     tbr[1] = new RoundedRectangle(Width / 2, 0, Width, 2 * Height / 5);
00255                     tbr[2] = new RoundedRectangle(0, 2 * Height / 5, Width / 2, 4 * Height / 5);
00256                     tbr[3] = new RoundedRectangle(Width / 2, 2 * Height / 5, Width, 4 * Height / 5);
00257                     tbr[4] = new RoundedRectangle(0, 4 * Height / 5, Width, Height);
00258                 }
00259                 else
00260                 {
00261                     tbr[0] = new RoundedRectangle(0, 0, 2 * Width / 5, Height / 2);
00262                     tbr[1] = new RoundedRectangle(0, Height / 2, 2 * Width / 5, Height);
00263                     tbr[2] = new RoundedRectangle(2 * Width / 5, 0, 4 * Width / 5, Height / 2);
00264                     tbr[3] = new RoundedRectangle(2 * Width / 5, Height / 2, 4 * Width / 5, Height);
00265                     tbr[4] = new RoundedRectangle(4 * Width / 5, 0, Width, Height);
00266                 }
00267             }
00268             else if (divisions == 7)
00269             {
00270                 if (isVertical)
00271                 {
00272                     tbr[0] = new RoundedRectangle(0, 0, Width / 2, 2 * Height / 7);
00273                     tbr[1] = new RoundedRectangle(Width / 2, 0, Width, 2 * Height / 7);
00274                     tbr[2] = new RoundedRectangle(0, 2 * Height / 7, Width / 2, 4 * Height / 7);
00275                     tbr[3] = new RoundedRectangle(Width / 2, 2 * Height / 7, Width, 4 * Height / 7);
00276                     tbr[4] = new RoundedRectangle(0, 4 * Height / 7, Width / 2, 6 * Height / 7);
00277                     tbr[5] = new RoundedRectangle(Width / 2, 4 * Height / 7, Width, 6 * Height / 7);
00278                     tbr[6] = new RoundedRectangle(0, 6 * Height / 7, Width, Height);
00279                 }
00280                 else
00281                 {
00282                     tbr[0] = new RoundedRectangle(0, 0, 2 * Width / 7, Height / 2);
00283                     tbr[1] = new RoundedRectangle(0, Height / 2, 2 * Width / 7, Height);
00284                     tbr[2] = new RoundedRectangle(2 * Width / 7, 0, 4 * Width / 7, Height / 2);
00285                     tbr[3] = new RoundedRectangle(2 * Width / 7, Height / 2, 4 * Width / 7, Height);
00286                     tbr[4] = new RoundedRectangle(4 * Width / 7, 0, 6 * Width / 7, Height / 2);
00287                     tbr[5] = new RoundedRectangle(4 * Width / 7, Height / 2, 6 * Width / 7, Height);
00288                     tbr[6] = new RoundedRectangle(6 * Width / 7, 0, Width, Height);
00289                 }
00290             }
00291             else
00292             {
00293                 for (int divisorInd = 0; divisorInd < Utils.AcceptableDivisors.Length; divisorInd++)
00294                 {
00295                     if (divisions % Utils.AcceptableDivisors[divisorInd] == 0)
00296                     {
00297                         RoundedRectangle[] largerDivisions = this.Split(divisions /
    Utils.AcceptableDivisors[divisorInd]);
00298
00299                         int pos = 0;
00300
00301                         for (int i = 0; i < largerDivisions.Length; i++)
00302                         {
00303                             RoundedSize s = new RoundedSize(largerDivisions[i].Width,
    largerDivisions[i].Height);
00304                             RoundedRectangle[] currDivision =
    s.Split(Utils.AcceptableDivisors[divisorInd]);
00305
00306                             for (int j = 0; j < currDivision.Length; j++)
00307                             {
00308                                 tbr[pos] = new RoundedRectangle(largerDivisions[i].X0 +
    currDivision[j].X0, largerDivisions[i].Y0 + currDivision[j].Y0, largerDivisions[i].X0 +
    currDivision[j].X1, largerDivisions[i].Y0 + currDivision[j].Y1);
00309                                 pos++;
00310                             }
00311                         }
00312
00313                         break;
00314                     }
00315                 }
00316             }
00317
```

```
00318             return tbr;
00319
00320         }
00321     }
00322
00323 /// <summary>
00324 /// Represents a rectangle.
00325 /// </summary>
00326     public struct Rectangle
00327     {
00328 /// <summary>
00329 /// The left coordinate of the rectangle.
00330 /// </summary>
00331         public float X0;
00332
00333 /// <summary>
00334 /// The top coordinate of the rectangle.
00335 /// </summary>
00336         public float Y0;
00337
00338 /// <summary>
00339 /// The right coordinate of the rectangle.
00340 /// </summary>
00341         public float X1;
00342
00343 /// <summary>
00344 /// The bottom coordinate of the rectangle.
00345 /// </summary>
00346         public float Y1;
00347
00348 /// <summary>
00349 /// The width of the rectangle.
00350 /// </summary>
00351         public float Width => X1 - X0;
00352
00353 /// <summary>
00354 /// The height of the rectangle.
00355 /// </summary>
00356         public float Height => Y1 - Y0;
00357
00358 /// <summary>
00359 /// Create a new <see cref="Rectangle"/> from the specified coordinates.
00360 /// </summary>
00361 /// <param name="x0">The left coordinate of the rectangle.</param>
00362 /// <param name="y0">The top coordinate of the rectangle.</param>
00363 /// <param name="x1">The right coordinate of the rectangle.</param>
00364 /// <param name="y1">The bottom coordinate of the rectangle.</param>
00365         public Rectangle(float x0, float y0, float x1, float y1)
00366         {
00367             X0 = x0;
00368             Y0 = y0;
00369             X1 = x1;
00370             Y1 = y1;
00371         }
00372
00373 /// <summary>
00374 /// Create a new <see cref="Rectangle"/> from the specified coordinates.
00375 /// </summary>
00376 /// <param name="x0">The left coordinate of the rectangle.</param>
00377 /// <param name="y0">The top coordinate of the rectangle.</param>
00378 /// <param name="x1">The right coordinate of the rectangle.</param>
00379 /// <param name="y1">The bottom coordinate of the rectangle.</param>
00380         public Rectangle(double x0, double y0, double x1, double y1)
00381         {
00382             X0 = (float)x0;
00383             Y0 = (float)y0;
00384             X1 = (float)x1;
00385             Y1 = (float)y1;
00386         }
00387
00388 /// <summary>
00389 /// Round the rectangle's coordinates to the closest integers.
00390 /// </summary>
00391 /// <returns>A <see cref="RoundedRectangle"/> with the rounded coordinates.</returns>
00392         public RoundedRectangle Round()
00393         {
00394             return new RoundedRectangle(
00395                 (int)Math.Floor(X0 + 0.001),
00396                 (int)Math.Floor(Y0 + 0.001),
00397                 (int)Math.Ceiling(X1 - 0.001),
00398                 (int)Math.Ceiling(Y1 - 0.001)
00399             );
00400         }
00401
00402 /// <summary>
00403 /// Round the rectangle's coordinates to the closest integers, applying the specified zoom factor.
00404 /// </summary>
```

```
00405 /// <param name="zoom">The zoom factor to apply.</param>
00406 /// <returns>A <see cref="RoundedRectangle"/> with the rounded coordinates.</returns>
00407         public RoundedRectangle Round(double zoom)
00408         {
00409             return new RoundedRectangle(
00410                (int)Math.Floor(X0 * (float)zoom + 0.001),
00411                (int)Math.Floor(Y0 * (float)zoom + 0.001),
00412                (int)Math.Ceiling(X1 * (float)zoom - 0.001),
00413                (int)Math.Ceiling(Y1 * (float)zoom - 0.001)
00414            );
00415         }
00416 
00417 /// <summary>
00418 /// Split the rectangle into the specified number of <see cref="Rectangle"/>s.
00419 /// </summary>
00420 /// <param name="divisions">The number of rectangles in which the rectangle should be split.  This
      must be factorisable using only powers of 2, 3, 5 or 7.  Otherwise, the biggest number smaller than
      <paramref name="divisions"/> that satisfies this condition is used.</param>
00421 /// <returns>An array of <see cref="Rectangle"/>s that when positioned properly cover the same area as
      this object.</returns>
00422         public Rectangle[] Split(int divisions)
00423         {
00424             Size s = new Size(this.Width, this.Height);
00425 
00426             Rectangle[] splitSize = s.Split(divisions);
00427 
00428             Rectangle[] tbr = new Rectangle[divisions];
00429 
00430             for (int i = 0; i < splitSize.Length; i++)
00431             {
00432                 tbr[i] = new Rectangle(this.X0 + splitSize[i].X0, this.Y0 + splitSize[i].Y0, this.X0 +
      splitSize[i].X1, this.Y0 + splitSize[i].Y1);
00433             }
00434 
00435             return tbr;
00436         }
00437 
00438 /// <summary>
00439 /// Compute the intersection between this <see cref="Rectangle"/> and another one.
00440 /// </summary>
00441 /// <param name="other">The other <see cref="Rectangle"/> to intersect with this instance.</param>
00442 /// <returns>The intersection between the two <see cref="Rectangle"/>s.</returns>
00443         public Rectangle Intersect(Rectangle other)
00444         {
00445             float x0 = Math.Max(this.X0, other.X0);
00446             float y0 = Math.Max(this.Y0, other.Y0);
00447 
00448             float x1 = Math.Min(this.X1, other.X1);
00449             float y1 = Math.Min(this.Y1, other.Y1);
00450 
00451             if (x1 <= x0 || y1 <= y0)
00452             {
00453                 return new Rectangle(0, 0, 0, 0);
00454             }
00455             else
00456             {
00457                 return new Rectangle(x0, y0, x1, y1);
00458             }
00459         }
00460 
00461 /// <summary>
00462 /// Checks whether this <see cref="Rectangle"/> contains another <see cref="Rectangle"/>.
00463 /// </summary>
00464 /// <param name="other">The <see cref="Rectangle"/> to check.</param>
00465 /// <returns>A boolean value indicating whether this <see cref="Rectangle"/> contains the <paramref
      name="other"/> <see cref="Rectangle"/>.</returns>
00466         public bool Contains(Rectangle other)
00467         {
00468             return other.X0 >= this.X0 && other.X1 <= this.X1 && other.Y0 >= this.Y0 && other.Y1 <=
      this.Y1;
00469         }
00470 
00471 /// <summary>
00472 /// Checks whether this <see cref="Rectangle"/> contains a <see cref="PointF"/>.
00473 /// </summary>
00474 /// <param name="point">The <see cref="PointF"/> to check.</param>
00475 /// <returns>A boolean value indicating whether this <see cref="Rectangle"/> contains the <paramref
      name="point"/>.</returns>
00476         public bool Contains(PointF point)
00477         {
00478             return point.X >= this.X0 && point.X <= this.X1 && point.Y >= this.Y0 && point.Y <=
      this.Y1;
00479         }
00480 
00481 /// <summary>
00482 /// Converts the <see cref="Rectangle"/> to a <see cref="Quad"/>.
00483 /// </summary>
```

```
00484 /// <returns>A <see cref="Quad"/> corresponding to the current <see cref="Rectangle"/>.</returns>
00485         public Quad ToQuad()
00486         {
00487             return new Quad(new PointF(X0, Y1), new PointF(X0, Y0), new PointF(X1, Y0), new PointF(X1,
      Y1));
00488         }
00489     }
00490 
00491 /// <summary>
00492 /// Represents a rectangle using only integer numbers.
00493 /// </summary>
00494     public struct RoundedRectangle
00495     {
00496 /// <summary>
00497 /// The left coordinate of the rectangle.
00498 /// </summary>
00499         public int X0;
00500 
00501 /// <summary>
00502 /// The top coordinate of the rectangle.
00503 /// </summary>
00504         public int Y0;
00505 
00506 /// <summary>
00507 /// The right coordinate of the rectangle.
00508 /// </summary>
00509         public int X1;
00510 
00511 /// <summary>
00512 /// The bottom coordinate of the rectangle.
00513 /// </summary>
00514         public int Y1;
00515 
00516 /// <summary>
00517 /// The width of the rectangle.
00518 /// </summary>
00519         public int Width => X1 - X0;
00520 
00521 /// <summary>
00522 /// The height of the rectangle.
00523 /// </summary>
00524         public int Height => Y1 - Y0;
00525 
00526 /// <summary>
00527 /// Create a new <see cref="RoundedRectangle"/> from the specified coordinates.
00528 /// </summary>
00529 /// <param name="x0">The left coordinate of the rectangle.</param>
00530 /// <param name="y0">The top coordinate of the rectangle.</param>
00531 /// <param name="x1">The right coordinate of the rectangle.</param>
00532 /// <param name="y1">The bottom coordinate of the rectangle.</param>
00533         public RoundedRectangle(int x0, int y0, int x1, int y1)
00534         {
00535             X0 = x0;
00536             Y0 = y0;
00537             X1 = x1;
00538             Y1 = y1;
00539         }
00540 
00541 /// <summary>
00542 /// Split the rectangle into the specified number of <see cref="RoundedRectangle"/>s.
00543 /// </summary>
00544 /// <param name="divisions">The number of rectangles in which the rectangle should be split.  This
      must be factorisable using only powers of 2, 3, 5 or 7.  Otherwise, the biggest number smaller than
      <paramref name="divisions"/> that satisfies this condition is used.</param>
00545 /// <returns>An array of <see cref="RoundedRectangle"/>s that when positioned properly cover the same
      area as this object.</returns>
00546         public RoundedRectangle[] Split(int divisions)
00547         {
00548             RoundedSize s = new RoundedSize(this.Width, this.Height);
00549 
00550             RoundedRectangle[] splitSize = s.Split(divisions);
00551 
00552             RoundedRectangle[] tbr = new RoundedRectangle[divisions];
00553 
00554             for (int i = 0; i < splitSize.Length; i++)
00555             {
00556                 tbr[i] = new RoundedRectangle(this.X0 + splitSize[i].X0, this.Y0 + splitSize[i].Y0,
      this.X0 + splitSize[i].X1, this.Y0 + splitSize[i].Y1);
00557             }
00558 
00559             return tbr;
00560         }
00561     }
00562 
00563 /// <summary>
00564 /// Represents a point.
00565 /// </summary>
```

```
00566     public struct PointF
00567     {
00568 /// <summary>
00569 /// The horizontal coordinate of the point.
00570 /// </summary>
00571         public float X;
00572
00573 /// <summary>
00574 /// The vertical coordinate of the point.
00575 /// </summary>
00576         public float Y;
00577
00578 /// <summary>
00579 /// Create a new <see cref="PointF"/> from the specified coordinates.
00580 /// </summary>
00581 /// <param name="x">The horizontal coordinate of the point.</param>
00582 /// <param name="y">The vertical coordinate of the point.</param>
00583         public PointF(float x, float y)
00584         {
00585             X = x;
00586             Y = y;
00587         }
00588     }
00589
00590 /// <summary>
00591 /// Represents a quadrilater (not necessarily a rectangle).
00592 /// </summary>
00593     public struct Quad
00594     {
00595 /// <summary>
00596 /// The lower left point of the quadrilater.
00597 /// </summary>
00598         public PointF LowerLeft;
00599
00600 /// <summary>
00601 /// The upper left point of the quadrilater.
00602 /// </summary>
00603         public PointF UpperLeft;
00604
00605 /// <summary>
00606 /// The upper right point of the quadrilater.
00607 /// </summary>
00608         public PointF UpperRight;
00609
00610 /// <summary>
00611 /// The lower right point of the quadrilater.
00612 /// </summary>
00613         public PointF LowerRight;
00614
00615 /// <summary>
00616 /// Creates a new <see cref="Quad"/> from the specified points.
00617 /// </summary>
00618 /// <param name="lowerLeft">The lower left point of the quadrilater.</param>
00619 /// <param name="upperLeft">The upper left point of the quadrilater.</param>
00620 /// <param name="upperRight">The upper right point of the quadrilater.</param>
00621 /// <param name="lowerRight">The lower right point of the quadrilater.</param>
00622         public Quad(PointF lowerLeft, PointF upperLeft, PointF upperRight, PointF lowerRight)
00623         {
00624             this.LowerLeft = lowerLeft;
00625             this.UpperLeft = upperLeft;
00626             this.UpperRight = upperRight;
00627             this.LowerRight = lowerRight;
00628         }
00629
00630 /// <summary>
00631 /// Checks whether this <see cref="Quad"/> contains a <see cref="PointF"/>.
00632 /// </summary>
00633 /// <param name="point">The <see cref="PointF"/> to check.</param>
00634 /// <returns>A boolean value indicating whether this <see cref="Quad"/> contains the <paramref
      name="point"/>.</returns>
00635         public bool Contains(PointF point)
00636         {
00637             return PointInTriangle(point, this.LowerLeft, this.UpperLeft, this.UpperRight) ||
      PointInTriangle(point, this.LowerLeft, this.UpperRight, this.LowerRight);
00638         }
00639
00640 /// <summary>
00641 /// Checks whether a point is contained in a triangle.
00642 /// </summary>
00643 /// <param name="pt">The point to test.</param>
00644 /// <param name="A">The first vertex of the triangle.</param>
00645 /// <param name="B">The second vertex of the triangle.</param>
00646 /// <param name="C">The third vertex of the triangle.</param>
00647 /// <returns>A boolean value indicating whether the point is contained in the triangle.</returns>
00648         private static bool PointInTriangle(PointF pt, PointF A, PointF B, PointF C)
00649         {
00650             double signAB = (pt.X - B.X) * (A.Y - B.Y) - (A.X - B.X) * (pt.Y - B.Y);
```

```
00651             double signBC = (pt.X - C.X) * (B.Y - C.Y) - (B.X - C.X) * (pt.Y - C.Y);
00652             double signCA = (pt.X - A.X) * (C.Y - A.Y) - (C.X - A.X) * (pt.Y - A.Y);
00653 
00654             return !((signAB < 0 || signBC < 0 || signCA < 0) && (signAB > 0 || signBC > 0 || signCA >
      0));
00655         }
00656     }
00657 }
```

8.12 TesseractLanguage.cs

```
00001 using System;
00002 using System.Collections.Generic;
00003 using System.IO;
00004 using System.Net;
00005 using System.Reflection;
00006 using System.Text;
00007 
00008 namespace MuPDFCore
00009 {
00010 /// <summary>
00011 /// Represents a language used by Tesseract OCR.
00012 /// </summary>
00013     public class TesseractLanguage
00014     {
00015 /// <summary>
00016 /// The name of the folder where the language file is located.
00017 /// </summary>
00018         public string Prefix { get; }
00019 
00020 /// <summary>
00021 /// The name of the language.  The Tesseract library will assume that the trained language data file
      can be found at <c>Prefix/Language.traineddata</c>.
00022 /// </summary>
00023         public string Language { get; }
00024 
00025 /// <summary>
00026 /// Fast integer versions of trained models.  These are models for a single language.
00027 /// </summary>
00028         public enum Fast
00029         {
00030 /// <summary>
00031 /// The Afrikaans language.
00032 /// </summary>
00033             Afr,
00034 /// <summary>
00035 /// The Amharic language.
00036 /// </summary>
00037             Amh,
00038 /// <summary>
00039 /// The Arabic language.
00040 /// </summary>
00041             Ara,
00042 /// <summary>
00043 /// The Assamese language.
00044 /// </summary>
00045             Asm,
00046 /// <summary>
00047 /// The Azerbaijani language.
00048 /// </summary>
00049             Aze,
00050 /// <summary>
00051 /// The Azerbaijani language (Cyrillic).
00052 /// </summary>
00053             Aze_Cyrl,
00054 /// <summary>
00055 /// The Belarusian language.
00056 /// </summary>
00057             Bel,
00058 /// <summary>
00059 /// The Bengali language.
00060 /// </summary>
00061             Ben,
00062 /// <summary>
00063 /// The Tibetan language.
00064 /// </summary>
00065             Bod,
00066 /// <summary>
00067 /// The Bosnian language.
00068 /// </summary>
00069             Bos,
00070 /// <summary>
00071 /// The Breton language.
```

```
00072 /// </summary>
00073             Bre,
00074 /// <summary>
00075 /// The Bulgarian language.
00076 /// </summary>
00077             Bul,
00078 /// <summary>
00079 /// The Catalan/Valencian language.
00080 /// </summary>
00081             Cat,
00082 /// <summary>
00083 /// The Cebuano language.
00084 /// </summary>
00085             Ceb,
00086 /// <summary>
00087 /// The Czech language.
00088 /// </summary>
00089             Ces,
00090 /// <summary>
00091 /// The Chinese (Simplified) language.
00092 /// </summary>
00093             Chi_Sim,
00094 /// <summary>
00095 /// The Chinese (Simplified) language (vertical).
00096 /// </summary>
00097             Chi_Sim_Vert,
00098 /// <summary>
00099 /// The Chinese (Traditional) language.
00100 /// </summary>
00101             Chi_Tra,
00102 /// <summary>
00103 /// The Chinese (Traditional) language (vertical).
00104 /// </summary>
00105             Chi_Tra_Vert,
00106 /// <summary>
00107 /// The Cherokee language.
00108 /// </summary>
00109             Chr,
00110 /// <summary>
00111 /// The Corsican language.
00112 /// </summary>
00113             Cos,
00114 /// <summary>
00115 /// The Welsh language.
00116 /// </summary>
00117             Cym,
00118 /// <summary>
00119 /// The Danish language.
00120 /// </summary>
00121             Dan,
00122 /// <summary>
00123 /// The German language.
00124 /// </summary>
00125             Deu,
00126 /// <summary>
00127 /// The Divehi/Dhivehi/Maldivian language.
00128 /// </summary>
00129             Div,
00130 /// <summary>
00131 /// The Dzongkha language.
00132 /// </summary>
00133             Dzo,
00134 /// <summary>
00135 /// The Greek, Modern (1453-) language.
00136 /// </summary>
00137             Ell,
00138 /// <summary>
00139 /// The English language.
00140 /// </summary>
00141             Eng,
00142 /// <summary>
00143 /// The English, Middle (1100-1500) language.
00144 /// </summary>
00145             Enm,
00146 /// <summary>
00147 /// The Esperanto language.
00148 /// </summary>
00149             Epo,
00150 /// <summary>
00151 /// A language for equations.
00152 /// </summary>
00153             Equ,
00154 /// <summary>
00155 /// The Estonian language.
00156 /// </summary>
00157             Est,
00158 /// <summary>
```

```
00159 /// The Basque language.
00160 /// </summary>
00161             Eus,
00162 /// <summary>
00163 /// The Faroese language.
00164 /// </summary>
00165             Fao,
00166 /// <summary>
00167 /// The Persian language.
00168 /// </summary>
00169             Fas,
00170 /// <summary>
00171 /// The Filipino/Pilipino language.
00172 /// </summary>
00173             Fil,
00174 /// <summary>
00175 /// The Finnish language.
00176 /// </summary>
00177             Fin,
00178 /// <summary>
00179 /// The French language.
00180 /// </summary>
00181             Fra,
00182 /// <summary>
00183 /// The German - Fraktur language.
00184 /// </summary>
00185             Frk,
00186 /// <summary>
00187 /// The French, Middle (ca.1400-1600) language.
00188 /// </summary>
00189             Frm,
00190 /// <summary>
00191 /// The Western Frisian language.
00192 /// </summary>
00193             Fry,
00194 /// <summary>
00195 /// The Gaelic/Scottish Gaelic language.
00196 /// </summary>
00197             Gla,
00198 /// <summary>
00199 /// The Irish language.
00200 /// </summary>
00201             Gle,
00202 /// <summary>
00203 /// The Galician language.
00204 /// </summary>
00205             Glg,
00206 /// <summary>
00207 /// The Greek, Ancient (to 1453) language.
00208 /// </summary>
00209             Grc,
00210 /// <summary>
00211 /// The Gujarati language.
00212 /// </summary>
00213             Guj,
00214 /// <summary>
00215 /// The Haitian/Haitian Creole language.
00216 /// </summary>
00217             Hat,
00218 /// <summary>
00219 /// The Hebrew language.
00220 /// </summary>
00221             Heb,
00222 /// <summary>
00223 /// The Hindi language.
00224 /// </summary>
00225             Hin,
00226 /// <summary>
00227 /// The Croatian language.
00228 /// </summary>
00229             Hrv,
00230 /// <summary>
00231 /// The Hungarian language.
00232 /// </summary>
00233             Hun,
00234 /// <summary>
00235 /// The Armenian language.
00236 /// </summary>
00237             Hye,
00238 /// <summary>
00239 /// The Inuktitut language.
00240 /// </summary>
00241             Iku,
00242 /// <summary>
00243 /// The Indonesian language.
00244 /// </summary>
00245             Ind,
```

```
00246 /// <summary>
00247 /// The Icelandic language.
00248 /// </summary>
00249             Isl,
00250 /// <summary>
00251 /// The Italian language.
00252 /// </summary>
00253             Ita,
00254 /// <summary>
00255 /// The Italian language (old).
00256 /// </summary>
00257             Ita_Old,
00258 /// <summary>
00259 /// The Javanese language.
00260 /// </summary>
00261             Jav,
00262 /// <summary>
00263 /// The Japanese language.
00264 /// </summary>
00265             Jpn,
00266 /// <summary>
00267 /// The Japanese language (vertical).
00268 /// </summary>
00269             Jpn_Vert,
00270 /// <summary>
00271 /// The Kannada language.
00272 /// </summary>
00273             Kan,
00274 /// <summary>
00275 /// The Georgian language.
00276 /// </summary>
00277             Kat,
00278 /// <summary>
00279 /// The Georgian language (old).
00280 /// </summary>
00281             Kat_Old,
00282 /// <summary>
00283 /// The Kazakh language.
00284 /// </summary>
00285             Kaz,
00286 /// <summary>
00287 /// The Central Khmer language.
00288 /// </summary>
00289             Khm,
00290 /// <summary>
00291 /// The Kirghiz/Kyrgyz language.
00292 /// </summary>
00293             Kir,
00294 /// <summary>
00295 /// The Northern Kurdish language.
00296 /// </summary>
00297             Kmr,
00298 /// <summary>
00299 /// The Korean language.
00300 /// </summary>
00301             Kor,
00302 /// <summary>
00303 /// The Korean language (vertical).
00304 /// </summary>
00305             Kor_Vert,
00306 /// <summary>
00307 /// The Lao language.
00308 /// </summary>
00309             Lao,
00310 /// <summary>
00311 /// The Latin language.
00312 /// </summary>
00313             Lat,
00314 /// <summary>
00315 /// The Latvian language.
00316 /// </summary>
00317             Lav,
00318 /// <summary>
00319 /// The Lithuanian language.
00320 /// </summary>
00321             Lit,
00322 /// <summary>
00323 /// The Luxembourgish/Letzeburgesch language.
00324 /// </summary>
00325             Ltz,
00326 /// <summary>
00327 /// The Malayalam language.
00328 /// </summary>
00329             Mal,
00330 /// <summary>
00331 /// The Marathi language.
00332 /// </summary>
```

```
00333             Mar,
00334 /// <summary>
00335 /// The Macedonian language.
00336 /// </summary>
00337             Mkd,
00338 /// <summary>
00339 /// The Maltese language.
00340 /// </summary>
00341             Mlt,
00342 /// <summary>
00343 /// The Mongolian language.
00344 /// </summary>
00345             Mon,
00346 /// <summary>
00347 /// The Maori language.
00348 /// </summary>
00349             Mri,
00350 /// <summary>
00351 /// The Malay language.
00352 /// </summary>
00353             Msa,
00354 /// <summary>
00355 /// The Burmese language.
00356 /// </summary>
00357             Mya,
00358 /// <summary>
00359 /// The Nepali language.
00360 /// </summary>
00361             Nep,
00362 /// <summary>
00363 /// The Dutch/Flemish language.
00364 /// </summary>
00365             Nld,
00366 /// <summary>
00367 /// The Norwegian language.
00368 /// </summary>
00369             Nor,
00370 /// <summary>
00371 /// The Occitan (post 1500) language.
00372 /// </summary>
00373             Oci,
00374 /// <summary>
00375 /// The Oriya language.
00376 /// </summary>
00377             Ori,
00378 /// <summary>
00379 /// The Orientation and script detection module.
00380 /// </summary>
00381             Osd,
00382 /// <summary>
00383 /// The Panjabi/Punjabi language.
00384 /// </summary>
00385             Pan,
00386 /// <summary>
00387 /// The Polish language.
00388 /// </summary>
00389             Pol,
00390 /// <summary>
00391 /// The Portuguese language.
00392 /// </summary>
00393             Por,
00394 /// <summary>
00395 /// The Pushto/Pashto language.
00396 /// </summary>
00397             Pus,
00398 /// <summary>
00399 /// The Quechua language.
00400 /// </summary>
00401             Que,
00402 /// <summary>
00403 /// The Romanian/Moldavian/Moldovan language.
00404 /// </summary>
00405             Ron,
00406 /// <summary>
00407 /// The Russian language.
00408 /// </summary>
00409             Rus,
00410 /// <summary>
00411 /// The Sanskrit language.
00412 /// </summary>
00413             San,
00414 /// <summary>
00415 /// The Sinhala/Sinhalese language.
00416 /// </summary>
00417             Sin,
00418 /// <summary>
00419 /// The Slovak language.
```

```
00420 /// </summary>
00421             Slk,
00422 /// <summary>
00423 /// The Slovenian language.
00424 /// </summary>
00425             Slv,
00426 /// <summary>
00427 /// The Sindhi language.
00428 /// </summary>
00429             Snd,
00430 /// <summary>
00431 /// The Spanish/Castilian language.
00432 /// </summary>
00433             Spa,
00434 /// <summary>
00435 /// The Spanish/Castilian language (old).
00436 /// </summary>
00437             Spa_Old,
00438 /// <summary>
00439 /// The Albanian language.
00440 /// </summary>
00441             Sqi,
00442 /// <summary>
00443 /// The Serbian language.
00444 /// </summary>
00445             Srp,
00446 /// <summary>
00447 /// The Serbian language (Latin).
00448 /// </summary>
00449             Srp_Latn,
00450 /// <summary>
00451 /// The Sundanese language.
00452 /// </summary>
00453             Sun,
00454 /// <summary>
00455 /// The Swahili language.
00456 /// </summary>
00457             Swa,
00458 /// <summary>
00459 /// The Swedish language.
00460 /// </summary>
00461             Swe,
00462 /// <summary>
00463 /// The Syriac language.
00464 /// </summary>
00465             Syr,
00466 /// <summary>
00467 /// The Tamil language.
00468 /// </summary>
00469             Tam,
00470 /// <summary>
00471 /// The Tatar language.
00472 /// </summary>
00473             Tat,
00474 /// <summary>
00475 /// The Telugu language.
00476 /// </summary>
00477             Tel,
00478 /// <summary>
00479 /// The Tajik language.
00480 /// </summary>
00481             Tgk,
00482 /// <summary>
00483 /// The Thai language.
00484 /// </summary>
00485             Tha,
00486 /// <summary>
00487 /// The Tigrinya language.
00488 /// </summary>
00489             Tir,
00490 /// <summary>
00491 /// The Tonga (Tonga Islands) language.
00492 /// </summary>
00493             Ton,
00494 /// <summary>
00495 /// The Turkish language.
00496 /// </summary>
00497             Tur,
00498 /// <summary>
00499 /// The Uighur/Uyghur language.
00500 /// </summary>
00501             Uig,
00502 /// <summary>
00503 /// The Ukrainian language.
00504 /// </summary>
00505             Ukr,
00506 /// <summary>
```

```
00507 /// The Urdu language.
00508 /// </summary>
00509             Urd,
00510 /// <summary>
00511 /// The Uzbek language.
00512 /// </summary>
00513             Uzb,
00514 /// <summary>
00515 /// The Uzbek language (Cyrillic).
00516 /// </summary>
00517             Uzb_Cyrl,
00518 /// <summary>
00519 /// The Vietnamese language.
00520 /// </summary>
00521             Vie,
00522 /// <summary>
00523 /// The Yiddish language.
00524 /// </summary>
00525             Yid,
00526 /// <summary>
00527 /// The Yoruba language.
00528 /// </summary>
00529             Yor
00530         }
00531
00532 /// <summary>
00533 /// Fast integer versions of trained models.  These are models for a single script supporting one or
      more languages.
00534 /// </summary>
00535         public enum FastScripts
00536         {
00537 /// <summary>
00538 /// The Arabic script.
00539 /// </summary>
00540             Arabic,
00541 /// <summary>
00542 /// The Armenian script.
00543 /// </summary>
00544             Armenian,
00545 /// <summary>
00546 /// The Bengali script.
00547 /// </summary>
00548             Bengali,
00549 /// <summary>
00550 /// The Canadian Aboriginal script.
00551 /// </summary>
00552             Canadian_Aboriginal,
00553 /// <summary>
00554 /// The Cherokee script.
00555 /// </summary>
00556             Cherokee,
00557 /// <summary>
00558 /// The Cyrillic script.
00559 /// </summary>
00560             Cyrillic,
00561 /// <summary>
00562 /// The Devanagari script.
00563 /// </summary>
00564             Devanagari,
00565 /// <summary>
00566 /// The Ethiopic script.
00567 /// </summary>
00568             Ethiopic,
00569 /// <summary>
00570 /// The Fraktur script.
00571 /// </summary>
00572             Fraktur,
00573 /// <summary>
00574 /// The Georgian script.
00575 /// </summary>
00576             Georgian,
00577 /// <summary>
00578 /// The Greek script.
00579 /// </summary>
00580             Greek,
00581 /// <summary>
00582 /// The Gujarati script.
00583 /// </summary>
00584             Gujarati,
00585 /// <summary>
00586 /// The Gurmukhi script.
00587 /// </summary>
00588             Gurmukhi,
00589 /// <summary>
00590 /// The Han (Simplified) script.
00591 /// </summary>
00592             HanS,
```

```
00593 /// <summary>
00594 /// The Han (Simplified) script.  (vertical)
00595 /// </summary>
00596             HanS_Vert,
00597 /// <summary>
00598 /// The Han (Traditional) script.
00599 /// </summary>
00600             HanT,
00601 /// <summary>
00602 /// The Han (Traditional) script.  (vertical)
00603 /// </summary>
00604             HanT_Vert,
00605 /// <summary>
00606 /// The Hangul script.
00607 /// </summary>
00608             Hangul,
00609 /// <summary>
00610 /// The Hangul script.  (vertical)
00611 /// </summary>
00612             Hangul_Vert,
00613 /// <summary>
00614 /// The Hebrew script.
00615 /// </summary>
00616             Hebrew,
00617 /// <summary>
00618 /// The Japanese script.
00619 /// </summary>
00620             Japanese,
00621 /// <summary>
00622 /// The Japanese script.  (vertical)
00623 /// </summary>
00624             Japanese_Vert,
00625 /// <summary>
00626 /// The Kannada script.
00627 /// </summary>
00628             Kannada,
00629 /// <summary>
00630 /// The Khmer script.
00631 /// </summary>
00632             Khmer,
00633 /// <summary>
00634 /// The Lao script.
00635 /// </summary>
00636             Lao,
00637 /// <summary>
00638 /// The Latin script.
00639 /// </summary>
00640             Latin,
00641 /// <summary>
00642 /// The Malayalam script.
00643 /// </summary>
00644             Malayalam,
00645 /// <summary>
00646 /// The Myanmar script.
00647 /// </summary>
00648             Myanmar,
00649 /// <summary>
00650 /// The Oriya script.
00651 /// </summary>
00652             Oriya,
00653 /// <summary>
00654 /// The Sinhala script.
00655 /// </summary>
00656             Sinhala,
00657 /// <summary>
00658 /// The Syriac script.
00659 /// </summary>
00660             Syriac,
00661 /// <summary>
00662 /// The Tamil script.
00663 /// </summary>
00664             Tamil,
00665 /// <summary>
00666 /// The Telugu script.
00667 /// </summary>
00668             Telugu,
00669 /// <summary>
00670 /// The Thaana script.
00671 /// </summary>
00672             Thaana,
00673 /// <summary>
00674 /// The Thai script.
00675 /// </summary>
00676             Thai,
00677 /// <summary>
00678 /// The Tibetan script.
00679 /// </summary>
```

```
00680             Tibetan,
00681 /// <summary>
00682 /// The Vietnamese script.
00683 /// </summary>
00684             Vietnamese
00685         }
00686 
00687 /// <summary>
00688 /// Best (most accurate) trained models.  These are models for a single language.
00689 /// </summary>
00690         public enum Best
00691         {
00692 /// <summary>
00693 /// The Afrikaans language.
00694 /// </summary>
00695             Afr,
00696 /// <summary>
00697 /// The Amharic language.
00698 /// </summary>
00699             Amh,
00700 /// <summary>
00701 /// The Arabic language.
00702 /// </summary>
00703             Ara,
00704 /// <summary>
00705 /// The Assamese language.
00706 /// </summary>
00707             Asm,
00708 /// <summary>
00709 /// The Azerbaijani language.
00710 /// </summary>
00711             Aze,
00712 /// <summary>
00713 /// The Azerbaijani language (Cyrillic).
00714 /// </summary>
00715             Aze_Cyrl,
00716 /// <summary>
00717 /// The Belarusian language.
00718 /// </summary>
00719             Bel,
00720 /// <summary>
00721 /// The Bengali language.
00722 /// </summary>
00723             Ben,
00724 /// <summary>
00725 /// The Tibetan language.
00726 /// </summary>
00727             Bod,
00728 /// <summary>
00729 /// The Bosnian language.
00730 /// </summary>
00731             Bos,
00732 /// <summary>
00733 /// The Breton language.
00734 /// </summary>
00735             Bre,
00736 /// <summary>
00737 /// The Bulgarian language.
00738 /// </summary>
00739             Bul,
00740 /// <summary>
00741 /// The Catalan/Valencian language.
00742 /// </summary>
00743             Cat,
00744 /// <summary>
00745 /// The Cebuano language.
00746 /// </summary>
00747             Ceb,
00748 /// <summary>
00749 /// The Czech language.
00750 /// </summary>
00751             Ces,
00752 /// <summary>
00753 /// The Chinese (Simplified) language.
00754 /// </summary>
00755             Chi_Sim,
00756 /// <summary>
00757 /// The Chinese (Simplified) language (vertical).
00758 /// </summary>
00759             Chi_Sim_Vert,
00760 /// <summary>
00761 /// The Chinese (Traditional) language.
00762 /// </summary>
00763             Chi_Tra,
00764 /// <summary>
00765 /// The Chinese (Traditional) language (vertical).
00766 /// </summary>
```

```
00767             Chi_Tra_Vert,
00768 /// <summary>
00769 /// The Cherokee language.
00770 /// </summary>
00771             Chr,
00772 /// <summary>
00773 /// The Corsican language.
00774 /// </summary>
00775             Cos,
00776 /// <summary>
00777 /// The Welsh language.
00778 /// </summary>
00779             Cym,
00780 /// <summary>
00781 /// The Danish language.
00782 /// </summary>
00783             Dan,
00784 /// <summary>
00785 /// The German language.
00786 /// </summary>
00787             Deu,
00788 /// <summary>
00789 /// The Divehi/Dhivehi/Maldivian language.
00790 /// </summary>
00791             Div,
00792 /// <summary>
00793 /// The Dzongkha language.
00794 /// </summary>
00795             Dzo,
00796 /// <summary>
00797 /// The Greek, Modern (1453-) language.
00798 /// </summary>
00799             Ell,
00800 /// <summary>
00801 /// The English language.
00802 /// </summary>
00803             Eng,
00804 /// <summary>
00805 /// The English, Middle (1100-1500) language.
00806 /// </summary>
00807             Enm,
00808 /// <summary>
00809 /// The Esperanto language.
00810 /// </summary>
00811             Epo,
00812 /// <summary>
00813 /// The Estonian language.
00814 /// </summary>
00815             Est,
00816 /// <summary>
00817 /// The Basque language.
00818 /// </summary>
00819             Eus,
00820 /// <summary>
00821 /// The Faroese language.
00822 /// </summary>
00823             Fao,
00824 /// <summary>
00825 /// The Persian language.
00826 /// </summary>
00827             Fas,
00828 /// <summary>
00829 /// The Filipino/Pilipino language.
00830 /// </summary>
00831             Fil,
00832 /// <summary>
00833 /// The Finnish language.
00834 /// </summary>
00835             Fin,
00836 /// <summary>
00837 /// The French language.
00838 /// </summary>
00839             Fra,
00840 /// <summary>
00841 /// The German - Fraktur language.
00842 /// </summary>
00843             Frk,
00844 /// <summary>
00845 /// The French, Middle (ca.1400-1600) language.
00846 /// </summary>
00847             Frm,
00848 /// <summary>
00849 /// The Western Frisian language.
00850 /// </summary>
00851             Fry,
00852 /// <summary>
00853 /// The Gaelic/Scottish Gaelic language.
```

```
00854 /// </summary>
00855             Gla,
00856 /// <summary>
00857 /// The Irish language.
00858 /// </summary>
00859             Gle,
00860 /// <summary>
00861 /// The Galician language.
00862 /// </summary>
00863             Glg,
00864 /// <summary>
00865 /// The Greek, Ancient (to 1453) language.
00866 /// </summary>
00867             Grc,
00868 /// <summary>
00869 /// The Gujarati language.
00870 /// </summary>
00871             Guj,
00872 /// <summary>
00873 /// The Haitian/Haitian Creole language.
00874 /// </summary>
00875             Hat,
00876 /// <summary>
00877 /// The Hebrew language.
00878 /// </summary>
00879             Heb,
00880 /// <summary>
00881 /// The Hindi language.
00882 /// </summary>
00883             Hin,
00884 /// <summary>
00885 /// The Croatian language.
00886 /// </summary>
00887             Hrv,
00888 /// <summary>
00889 /// The Hungarian language.
00890 /// </summary>
00891             Hun,
00892 /// <summary>
00893 /// The Armenian language.
00894 /// </summary>
00895             Hye,
00896 /// <summary>
00897 /// The Inuktitut language.
00898 /// </summary>
00899             Iku,
00900 /// <summary>
00901 /// The Indonesian language.
00902 /// </summary>
00903             Ind,
00904 /// <summary>
00905 /// The Icelandic language.
00906 /// </summary>
00907             Isl,
00908 /// <summary>
00909 /// The Italian language.
00910 /// </summary>
00911             Ita,
00912 /// <summary>
00913 /// The Italian language (old).
00914 /// </summary>
00915             Ita_Old,
00916 /// <summary>
00917 /// The Javanese language.
00918 /// </summary>
00919             Jav,
00920 /// <summary>
00921 /// The Japanese language.
00922 /// </summary>
00923             Jpn,
00924 /// <summary>
00925 /// The Japanese language (vertical).
00926 /// </summary>
00927             Jpn_Vert,
00928 /// <summary>
00929 /// The Kannada language.
00930 /// </summary>
00931             Kan,
00932 /// <summary>
00933 /// The Georgian language.
00934 /// </summary>
00935             Kat,
00936 /// <summary>
00937 /// The Georgian language (old).
00938 /// </summary>
00939             Kat_Old,
00940 /// <summary>
```

```
00941 /// The Kazakh language.
00942 /// </summary>
00943             Kaz,
00944 /// <summary>
00945 /// The Central Khmer language.
00946 /// </summary>
00947             Khm,
00948 /// <summary>
00949 /// The Kirghiz/Kyrgyz language.
00950 /// </summary>
00951             Kir,
00952 /// <summary>
00953 /// The Northern Kurdish language.
00954 /// </summary>
00955             Kmr,
00956 /// <summary>
00957 /// The Korean language.
00958 /// </summary>
00959             Kor,
00960 /// <summary>
00961 /// The Korean language (vertical).
00962 /// </summary>
00963             Kor_Vert,
00964 /// <summary>
00965 /// The Lao language.
00966 /// </summary>
00967             Lao,
00968 /// <summary>
00969 /// The Latin language.
00970 /// </summary>
00971             Lat,
00972 /// <summary>
00973 /// The Latvian language.
00974 /// </summary>
00975             Lav,
00976 /// <summary>
00977 /// The Lithuanian language.
00978 /// </summary>
00979             Lit,
00980 /// <summary>
00981 /// The Luxembourgish/Letzeburgesch language.
00982 /// </summary>
00983             Ltz,
00984 /// <summary>
00985 /// The Malayalam language.
00986 /// </summary>
00987             Mal,
00988 /// <summary>
00989 /// The Marathi language.
00990 /// </summary>
00991             Mar,
00992 /// <summary>
00993 /// The Macedonian language.
00994 /// </summary>
00995             Mkd,
00996 /// <summary>
00997 /// The Maltese language.
00998 /// </summary>
00999             Mlt,
01000 /// <summary>
01001 /// The Mongolian language.
01002 /// </summary>
01003             Mon,
01004 /// <summary>
01005 /// The Maori language.
01006 /// </summary>
01007             Mri,
01008 /// <summary>
01009 /// The Malay language.
01010 /// </summary>
01011             Msa,
01012 /// <summary>
01013 /// The Burmese language.
01014 /// </summary>
01015             Mya,
01016 /// <summary>
01017 /// The Nepali language.
01018 /// </summary>
01019             Nep,
01020 /// <summary>
01021 /// The Dutch/Flemish language.
01022 /// </summary>
01023             Nld,
01024 /// <summary>
01025 /// The Norwegian language.
01026 /// </summary>
01027             Nor,
```

```
01028 /// <summary>
01029 /// The Occitan (post 1500) language.
01030 /// </summary>
01031             Oci,
01032 /// <summary>
01033 /// The Oriya language.
01034 /// </summary>
01035             Ori,
01036 /// <summary>
01037 /// The Orientation and script detection module.
01038 /// </summary>
01039             Osd,
01040 /// <summary>
01041 /// The Panjabi/Punjabi language.
01042 /// </summary>
01043             Pan,
01044 /// <summary>
01045 /// The Polish language.
01046 /// </summary>
01047             Pol,
01048 /// <summary>
01049 /// The Portuguese language.
01050 /// </summary>
01051             Por,
01052 /// <summary>
01053 /// The Pushto/Pashto language.
01054 /// </summary>
01055             Pus,
01056 /// <summary>
01057 /// The Quechua language.
01058 /// </summary>
01059             Que,
01060 /// <summary>
01061 /// The Romanian/Moldavian/Moldovan language.
01062 /// </summary>
01063             Ron,
01064 /// <summary>
01065 /// The Russian language.
01066 /// </summary>
01067             Rus,
01068 /// <summary>
01069 /// The Sanskrit language.
01070 /// </summary>
01071             San,
01072 /// <summary>
01073 /// The Sinhala/Sinhalese language.
01074 /// </summary>
01075             Sin,
01076 /// <summary>
01077 /// The Slovak language.
01078 /// </summary>
01079             Slk,
01080 /// <summary>
01081 /// The Slovenian language.
01082 /// </summary>
01083             Slv,
01084 /// <summary>
01085 /// The Sindhi language.
01086 /// </summary>
01087             Snd,
01088 /// <summary>
01089 /// The Spanish/Castilian language.
01090 /// </summary>
01091             Spa,
01092 /// <summary>
01093 /// The Spanish/Castilian language (old).
01094 /// </summary>
01095             Spa_Old,
01096 /// <summary>
01097 /// The Albanian language.
01098 /// </summary>
01099             Sqi,
01100 /// <summary>
01101 /// The Serbian language.
01102 /// </summary>
01103             Srp,
01104 /// <summary>
01105 /// The Serbian language (Latin).
01106 /// </summary>
01107             Srp_Latn,
01108 /// <summary>
01109 /// The Sundanese language.
01110 /// </summary>
01111             Sun,
01112 /// <summary>
01113 /// The Swahili language.
01114 /// </summary>
```

```
01115             Swa,
01116 /// <summary>
01117 /// The Swedish language.
01118 /// </summary>
01119             Swe,
01120 /// <summary>
01121 /// The Syriac language.
01122 /// </summary>
01123             Syr,
01124 /// <summary>
01125 /// The Tamil language.
01126 /// </summary>
01127             Tam,
01128 /// <summary>
01129 /// The Tatar language.
01130 /// </summary>
01131             Tat,
01132 /// <summary>
01133 /// The Telugu language.
01134 /// </summary>
01135             Tel,
01136 /// <summary>
01137 /// The Tajik language.
01138 /// </summary>
01139             Tgk,
01140 /// <summary>
01141 /// The Thai language.
01142 /// </summary>
01143             Tha,
01144 /// <summary>
01145 /// The Tigrinya language.
01146 /// </summary>
01147             Tir,
01148 /// <summary>
01149 /// The Tonga (Tonga Islands) language.
01150 /// </summary>
01151             Ton,
01152 /// <summary>
01153 /// The Turkish language.
01154 /// </summary>
01155             Tur,
01156 /// <summary>
01157 /// The Uighur/Uyghur language.
01158 /// </summary>
01159             Uig,
01160 /// <summary>
01161 /// The Ukrainian language.
01162 /// </summary>
01163             Ukr,
01164 /// <summary>
01165 /// The Urdu language.
01166 /// </summary>
01167             Urd,
01168 /// <summary>
01169 /// The Uzbek language.
01170 /// </summary>
01171             Uzb,
01172 /// <summary>
01173 /// The Uzbek language (Cyrillic).
01174 /// </summary>
01175             Uzb_Cyrl,
01176 /// <summary>
01177 /// The Vietnamese language.
01178 /// </summary>
01179             Vie,
01180 /// <summary>
01181 /// The Yiddish language.
01182 /// </summary>
01183             Yid,
01184 /// <summary>
01185 /// The Yoruba language.
01186 /// </summary>
01187             Yor
01188         }
01189
01190 /// <summary>
01191 /// Best (most accurate) trained models.  These are models for a single script supporting one or more
      languages.
01192 /// </summary>
01193         public enum BestScripts
01194         {
01195 /// <summary>
01196 /// The Arabic script.
01197 /// </summary>
01198             Arabic,
01199 /// <summary>
01200 /// The Armenian script.
```

```
01201 /// </summary>
01202             Armenian,
01203 /// <summary>
01204 /// The Bengali script.
01205 /// </summary>
01206             Bengali,
01207 /// <summary>
01208 /// The Canadian Aboriginal script.
01209 /// </summary>
01210             Canadian_Aboriginal,
01211 /// <summary>
01212 /// The Cherokee script.
01213 /// </summary>
01214             Cherokee,
01215 /// <summary>
01216 /// The Cyrillic script.
01217 /// </summary>
01218             Cyrillic,
01219 /// <summary>
01220 /// The Devanagari script.
01221 /// </summary>
01222             Devanagari,
01223 /// <summary>
01224 /// The Ethiopic script.
01225 /// </summary>
01226             Ethiopic,
01227 /// <summary>
01228 /// The Fraktur script.
01229 /// </summary>
01230             Fraktur,
01231 /// <summary>
01232 /// The Georgian script.
01233 /// </summary>
01234             Georgian,
01235 /// <summary>
01236 /// The Greek script.
01237 /// </summary>
01238             Greek,
01239 /// <summary>
01240 /// The Gujarati script.
01241 /// </summary>
01242             Gujarati,
01243 /// <summary>
01244 /// The Gurmukhi script.
01245 /// </summary>
01246             Gurmukhi,
01247 /// <summary>
01248 /// The Han (Simplified) script.
01249 /// </summary>
01250             HanS,
01251 /// <summary>
01252 /// The Han (Simplified) script.  (vertical)
01253 /// </summary>
01254             HanS_Vert,
01255 /// <summary>
01256 /// The Han (Traditional) script.
01257 /// </summary>
01258             HanT,
01259 /// <summary>
01260 /// The Han (Traditional) script.  (vertical)
01261 /// </summary>
01262             HanT_Vert,
01263 /// <summary>
01264 /// The Hangul script.
01265 /// </summary>
01266             Hangul,
01267 /// <summary>
01268 /// The Hangul script.  (vertical)
01269 /// </summary>
01270             Hangul_Vert,
01271 /// <summary>
01272 /// The Hebrew script.
01273 /// </summary>
01274             Hebrew,
01275 /// <summary>
01276 /// The Japanese script.
01277 /// </summary>
01278             Japanese,
01279 /// <summary>
01280 /// The Japanese script.  (vertical)
01281 /// </summary>
01282             Japanese_Vert,
01283 /// <summary>
01284 /// The Kannada script.
01285 /// </summary>
01286             Kannada,
01287 /// <summary>
```

```
01288 /// The Khmer script.
01289 /// </summary>
01290             Khmer,
01291 /// <summary>
01292 /// The Lao script.
01293 /// </summary>
01294             Lao,
01295 /// <summary>
01296 /// The Latin script.
01297 /// </summary>
01298             Latin,
01299 /// <summary>
01300 /// The Malayalam script.
01301 /// </summary>
01302             Malayalam,
01303 /// <summary>
01304 /// The Myanmar script.
01305 /// </summary>
01306             Myanmar,
01307 /// <summary>
01308 /// The Oriya script.
01309 /// </summary>
01310             Oriya,
01311 /// <summary>
01312 /// The Sinhala script.
01313 /// </summary>
01314             Sinhala,
01315 /// <summary>
01316 /// The Syriac script.
01317 /// </summary>
01318             Syriac,
01319 /// <summary>
01320 /// The Tamil script.
01321 /// </summary>
01322             Tamil,
01323 /// <summary>
01324 /// The Telugu script.
01325 /// </summary>
01326             Telugu,
01327 /// <summary>
01328 /// The Thaana script.
01329 /// </summary>
01330             Thaana,
01331 /// <summary>
01332 /// The Thai script.
01333 /// </summary>
01334             Thai,
01335 /// <summary>
01336 /// The Tibetan script.
01337 /// </summary>
01338             Tibetan,
01339 /// <summary>
01340 /// The Vietnamese script.
01341 /// </summary>
01342             Vietnamese
01343         }
01344
01345 /// <summary>
01346 /// Create a new <see cref="TesseractLanguage"/> object using the provided <paramref name="prefix"/>
      and <paramref name="language"/> name, without processing them in any way.
01347 /// </summary>
01348 /// <param name="prefix">The name of the folder where the language file is located.  If this is <see
      langword="null" />, the value of the environment variable <c>TESSDATA_PREFIX</c> will be used.</param>
01349 /// <param name="language">The name of the language.  The Tesseract library will assume that the
      trained language data file can be found at <paramref name="prefix"/><c>/</c><paramref
      name="language"/><c>.traineddata</c>.</param>
01350         public TesseractLanguage(string prefix, string language)
01351         {
01352             this.Prefix = prefix;
01353             this.Language = language;
01354         }
01355
01356 /// <summary>
01357 /// Create a new <see cref="TesseractLanguage"/> object using the specified trained model data file.
01358 /// </summary>
01359 /// <param name="fileName">The path to the trained model data file.  If the file name does not end in
      <c>.traineddata</c>, the file is copied to a temporary folder, and the temporary file is used by the
      Tesseract library.</param>
01360         public TesseractLanguage(string fileName)
01361         {
01362             if (fileName.EndsWith(".traineddata"))
01363             {
01364                 fileName = Path.GetFullPath(fileName);
01365
01366                 this.Prefix = Path.GetDirectoryName(fileName);
01367                 this.Language = Path.GetFileName(fileName).Substring(0,
      Path.GetFileName(fileName).Length - 12);
```

```
01368             }
01369             else
01370             {
01371                 this.Prefix = Path.GetTempPath();
01372                 this.Language = Guid.NewGuid().ToString("N");
01373 
01374                 File.Copy(fileName, Path.Combine(this.Prefix, this.Language + ".traineddata"));
01375             }
01376 
01377         }
01378 
01379         private static readonly string ExecutablePath =
      Path.GetDirectoryName(Assembly.GetEntryAssembly().Location);
01380         private static readonly string LocalCachePath =
      Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData);
01381 
01382 /// <summary>
01383 /// Create a new <see cref="TesseractLanguage"/> object using a fast integer version of a trained
      model for the specified language.  The language file is downloaded from the
      <c>tesseract-ocr/tessdata_fast</c> GitHub repository.  If it has already been downloaded and cached
      before, the downloaded file is re-used.
01384 /// </summary>
01385 /// <param name="language">The language to use for the OCR process.</param>
01386 /// <param name="useAnyCached">If this is <see langword="true"/>, if a cached trained model file is
      available for the specified language, it will be used even if it is a "best (most accurate)" model.
      Otherwise, only cached fast integer trained models will be used.</param>
01387         public TesseractLanguage(Fast language, bool useAnyCached = false)
01388         {
01389             string languageName = language.ToString().ToLower();
01390 
01391             string prefix = null;
01392 
01393             if (!string.IsNullOrEmpty(ExecutablePath) && File.Exists(Path.Combine(ExecutablePath,
      "tessdata", "fast", languageName + ".traineddata")))
01394             {
01395                 prefix = Path.Combine(ExecutablePath, "tessdata", "fast");
01396             }
01397             else if (!string.IsNullOrEmpty(ExecutablePath) && File.Exists(Path.Combine(ExecutablePath,
      "fast", languageName + ".traineddata")))
01398             {
01399                 prefix = Path.Combine(ExecutablePath, "fast");
01400             }
01401             else if (File.Exists(Path.Combine(LocalCachePath, "tessdata", "fast", languageName +
      ".traineddata")))
01402             {
01403                 prefix = Path.Combine(LocalCachePath, "tessdata", "fast");
01404             }
01405             else if (useAnyCached)
01406             {
01407                 if (!string.IsNullOrEmpty(ExecutablePath) && File.Exists(Path.Combine(ExecutablePath,
      languageName + ".traineddata")))
01408                 {
01409                     prefix = Path.Combine(ExecutablePath);
01410                 }
01411                 else if (!string.IsNullOrEmpty(ExecutablePath) &&
      File.Exists(Path.Combine(ExecutablePath, "tessdata", "best", languageName + ".traineddata")))
01412                 {
01413                     prefix = Path.Combine(ExecutablePath, "tessdata", "best");
01414                 }
01415                 else if (!string.IsNullOrEmpty(ExecutablePath) &&
      File.Exists(Path.Combine(ExecutablePath, "best", languageName + ".traineddata")))
01416                 {
01417                     prefix = Path.Combine(ExecutablePath, "best");
01418                 }
01419                 else if (File.Exists(Path.Combine(LocalCachePath, "tessdata", "best", languageName +
      ".traineddata")))
01420                 {
01421                     prefix = Path.Combine(LocalCachePath, "tessdata", "best");
01422                 }
01423             }
01424 
01425             if (prefix == null)
01426             {
01427                 string remotePath = "https://github.com/tesseract-ocr/tessdata_fast/raw/main/" +
      languageName + ".traineddata";
01428 
01429                 string localDirectory = Path.Combine(LocalCachePath, "tessdata", "fast");
01430 
01431                 if (!Directory.Exists(localDirectory))
01432                 {
01433                     Directory.CreateDirectory(localDirectory);
01434                 }
01435 
01436                 using (WebClient client = new WebClient())
01437                 {
01438                     client.DownloadFile(remotePath, Path.Combine(localDirectory, languageName +
      ".traineddata"));
```

```
01439                 }
01440
01441                 prefix = localDirectory;
01442             }
01443
01444             this.Prefix = prefix;
01445             this.Language = languageName;
01446         }
01447
01448 /// <summary>
01449 /// Create a new <see cref="TesseractLanguage"/> object using the best (most accurate) version of the
      trained model for the specified language.  The language file is downloaded from the
      <c>tesseract-ocr/tessdata_best</c> GitHub repository.  If it has already been downloaded and cached
      before, the downloaded file is re-used.
01450 /// </summary>
01451 /// <param name="language">The language to use for the OCR process.</param>
01452 /// <param name="useAnyCached">If this is <see langword="true"/>, if a cached trained model file is
      available for the specified language, it will be used even if it is a "fast" model.  Otherwise, only
      cached best (most accurate) trained models will be used.</param>
01453         public TesseractLanguage(Best language, bool useAnyCached = false)
01454         {
01455             string languageName = language.ToString().ToLower();
01456
01457             string prefix = null;
01458
01459             if (!string.IsNullOrEmpty(ExecutablePath) && File.Exists(Path.Combine(ExecutablePath,
      "tessdata", "best", languageName + ".traineddata")))
01460             {
01461                 prefix = Path.Combine(ExecutablePath, "tessdata", "best");
01462             }
01463             else if (!string.IsNullOrEmpty(ExecutablePath) && File.Exists(Path.Combine(ExecutablePath,
      "best", languageName + ".traineddata")))
01464             {
01465                 prefix = Path.Combine(ExecutablePath, "best");
01466             }
01467             else if (File.Exists(Path.Combine(LocalCachePath, "tessdata", "best", languageName +
      ".traineddata")))
01468             {
01469                 prefix = Path.Combine(LocalCachePath, "tessdata", "best");
01470             }
01471             else if (useAnyCached)
01472             {
01473                 if (!string.IsNullOrEmpty(ExecutablePath) && File.Exists(Path.Combine(ExecutablePath,
      languageName + ".traineddata")))
01474                 {
01475                     prefix = Path.Combine(ExecutablePath);
01476                 }
01477                 else if (!string.IsNullOrEmpty(ExecutablePath) &&
      File.Exists(Path.Combine(ExecutablePath, "tessdata", "fast", languageName + ".traineddata")))
01478                 {
01479                     prefix = Path.Combine(ExecutablePath, "tessdata", "fast");
01480                 }
01481                 else if (!string.IsNullOrEmpty(ExecutablePath) &&
      File.Exists(Path.Combine(ExecutablePath, "fast", languageName + ".traineddata")))
01482                 {
01483                     prefix = Path.Combine(ExecutablePath, "fast");
01484                 }
01485                 else if (File.Exists(Path.Combine(LocalCachePath, "tessdata", "fast", languageName +
      ".traineddata")))
01486                 {
01487                     prefix = Path.Combine(LocalCachePath, "tessdata", "fast");
01488                 }
01489             }
01490
01491             if (prefix == null)
01492             {
01493                 string remotePath = "https://github.com/tesseract-ocr/tessdata_best/raw/main/" +
      languageName + ".traineddata";
01494
01495                 string localDirectory = Path.Combine(LocalCachePath, "tessdata", "best");
01496
01497                 if (!Directory.Exists(localDirectory))
01498                 {
01499                     Directory.CreateDirectory(localDirectory);
01500                 }
01501
01502                 using (WebClient client = new WebClient())
01503                 {
01504                     client.DownloadFile(remotePath, Path.Combine(localDirectory, languageName +
      ".traineddata"));
01505                 }
01506
01507                 prefix = localDirectory;
01508             }
01509
01510             this.Prefix = prefix;
01511             this.Language = languageName;
```

```
01512         }
01513
01514 /// <summary>
01515 /// Create a new <see cref="TesseractLanguage"/> object using a fast integer version of a trained
      model for the specified script.  The language file is downloaded from the
      <c>tesseract-ocr/tessdata_fast</c> GitHub repository.  If it has already been downloaded and cached
      before, the downloaded file is re-used.
01516 /// </summary>
01517 /// <param name="script">The script to use for the OCR process.</param>
01518 /// <param name="useAnyCached">If this is <see langword="true"/>, if a cached trained model file is
      available for the specified script, it will be used even if it is a "best (most accurate)" model.
      Otherwise, only cached fast integer trained models will be used.</param>
01519         public TesseractLanguage(FastScripts script, bool useAnyCached = false)
01520         {
01521             string languageName = script.ToString().Replace("_Vert", "_vert");
01522
01523             string prefix = null;
01524
01525             if (!string.IsNullOrEmpty(ExecutablePath) && File.Exists(Path.Combine(ExecutablePath,
      "tessdata", "fast", "script", languageName + ".traineddata")))
01526             {
01527                 prefix = Path.Combine(ExecutablePath, "tessdata", "fast", "script");
01528             }
01529             else if (!string.IsNullOrEmpty(ExecutablePath) && File.Exists(Path.Combine(ExecutablePath,
      "fast", "script", languageName + ".traineddata")))
01530             {
01531                 prefix = Path.Combine(ExecutablePath, "fast", "script");
01532             }
01533             else if (File.Exists(Path.Combine(LocalCachePath, "tessdata", "fast", "script",
      languageName + ".traineddata")))
01534             {
01535                 prefix = Path.Combine(LocalCachePath, "tessdata", "fast", "script");
01536             }
01537             else if (useAnyCached)
01538             {
01539                 if (!string.IsNullOrEmpty(ExecutablePath) && File.Exists(Path.Combine(ExecutablePath,
      "script", languageName + ".traineddata")))
01540                 {
01541                     prefix = Path.Combine(ExecutablePath, "script");
01542                 }
01543                 else if (!string.IsNullOrEmpty(ExecutablePath) &&
      File.Exists(Path.Combine(ExecutablePath, languageName + ".traineddata")))
01544                 {
01545                     prefix = Path.Combine(ExecutablePath);
01546                 }
01547                 else if (!string.IsNullOrEmpty(ExecutablePath) &&
      File.Exists(Path.Combine(ExecutablePath, "tessdata", "best", "script", languageName +
      ".traineddata")))
01548                 {
01549                     prefix = Path.Combine(ExecutablePath, "tessdata", "best", "script");
01550                 }
01551                 else if (!string.IsNullOrEmpty(ExecutablePath) &&
      File.Exists(Path.Combine(ExecutablePath, "best", "script", languageName + ".traineddata")))
01552                 {
01553                     prefix = Path.Combine(ExecutablePath, "best", "script");
01554                 }
01555                 else if (File.Exists(Path.Combine(LocalCachePath, "tessdata", "best", "script",
      languageName + ".traineddata")))
01556                 {
01557                     prefix = Path.Combine(LocalCachePath, "tessdata", "best", "script");
01558                 }
01559             }
01560
01561             if (prefix == null)
01562             {
01563                 string remotePath = "https://github.com/tesseract-ocr/tessdata_fast/raw/main/script/"
      + languageName + ".traineddata";
01564
01565                 string localDirectory = Path.Combine(LocalCachePath, "tessdata", "fast", "script");
01566
01567                 if (!Directory.Exists(localDirectory))
01568                 {
01569                     Directory.CreateDirectory(localDirectory);
01570                 }
01571
01572                 using (WebClient client = new WebClient())
01573                 {
01574                     client.DownloadFile(remotePath, Path.Combine(localDirectory, languageName +
      ".traineddata"));
01575                 }
01576
01577                 prefix = localDirectory;
01578             }
01579
01580             this.Prefix = prefix;
01581             this.Language = languageName;
01582         }
```

```
01583
01584 /// <summary>
01585 /// Create a new <see cref="TesseractLanguage"/> object using the best (most accurate) version of the
      trained model for the specified script.  The language file is downloaded from the
      <c>tesseract-ocr/tessdata_best</c> GitHub repository.  If it has already been downloaded and cached
      before, the downloaded file is re-used.
01586 /// </summary>
01587 /// <param name="script">The script to use for the OCR process.</param>
01588 /// <param name="useAnyCached">If this is <see langword="true"/>, if a cached trained model file is
      available for the specified script, it will be used even if it is a "fast" model.  Otherwise, only
      cached best (most accurate) trained models will be used.</param>
01589         public TesseractLanguage(BestScripts script, bool useAnyCached = false)
01590         {
01591             string languageName = script.ToString().Replace("_Vert", "_vert");
01592
01593             string prefix = null;
01594
01595             if (!string.IsNullOrEmpty(ExecutablePath) && File.Exists(Path.Combine(ExecutablePath,
      "tessdata", "best", "script", languageName + ".traineddata")))
01596             {
01597                 prefix = Path.Combine(ExecutablePath, "tessdata", "best", "script");
01598             }
01599             else if (!string.IsNullOrEmpty(ExecutablePath) && File.Exists(Path.Combine(ExecutablePath,
      "best", "script", languageName + ".traineddata")))
01600             {
01601                 prefix = Path.Combine(ExecutablePath, "best", "script");
01602             }
01603             else if (File.Exists(Path.Combine(LocalCachePath, "tessdata", "best", "script",
      languageName + ".traineddata")))
01604             {
01605                 prefix = Path.Combine(LocalCachePath, "tessdata", "best", "script");
01606             }
01607             else if (useAnyCached)
01608             {
01609                 if (!string.IsNullOrEmpty(ExecutablePath) && File.Exists(Path.Combine(ExecutablePath,
      "script", languageName + ".traineddata")))
01610                 {
01611                     prefix = Path.Combine(ExecutablePath, "script");
01612                 }
01613                 else if (!string.IsNullOrEmpty(ExecutablePath) &&
      File.Exists(Path.Combine(ExecutablePath, languageName + ".traineddata")))
01614                 {
01615                     prefix = Path.Combine(ExecutablePath);
01616                 }
01617                 else if (!string.IsNullOrEmpty(ExecutablePath) &&
      File.Exists(Path.Combine(ExecutablePath, "tessdata", "fast", "script", languageName +
      ".traineddata")))
01618                 {
01619                     prefix = Path.Combine(ExecutablePath, "tessdata", "fast", "script");
01620                 }
01621                 else if (!string.IsNullOrEmpty(ExecutablePath) &&
      File.Exists(Path.Combine(ExecutablePath, "fast", "script", languageName + ".traineddata")))
01622                 {
01623                     prefix = Path.Combine(ExecutablePath, "fast", "script");
01624                 }
01625                 else if (File.Exists(Path.Combine(LocalCachePath, "tessdata", "fast", "script",
      languageName + ".traineddata")))
01626                 {
01627                     prefix = Path.Combine(LocalCachePath, "tessdata", "fast", "script");
01628                 }
01629             }
01630
01631             if (prefix == null)
01632             {
01633                 string remotePath = "https://github.com/tesseract-ocr/tessdata_best/raw/main/script/"
      + languageName + ".traineddata";
01634
01635                 string localDirectory = Path.Combine(LocalCachePath, "tessdata", "best", "script");
01636
01637                 if (!Directory.Exists(localDirectory))
01638                 {
01639                     Directory.CreateDirectory(localDirectory);
01640                 }
01641
01642                 using (WebClient client = new WebClient())
01643                 {
01644                     client.DownloadFile(remotePath, Path.Combine(localDirectory, languageName +
      ".traineddata"));
01645                 }
01646
01647                 prefix = localDirectory;
01648             }
01649
01650             this.Prefix = prefix;
01651             this.Language = languageName;
01652         }
01653     }
```

```
01654 }
```

8.13 Utils.cs

```
00001 /*
00002 MuPDFCore - A set of multiplatform .NET Core bindings for MuPDF.
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 System;
00019
00020 namespace MuPDFCore
00021 {
00022 /// <summary>
00023 /// Contains useful methods.
00024 /// </summary>
00025     internal static class Utils
00026     {
00027 /// <summary>
00028 /// The factors by which we can divide a rectangle/size.
00029 /// </summary>
00030         public static readonly int[] AcceptableDivisors = new[] { 2, 3, 5, 7 };
00031
00032 /// <summary>
00033 /// Computes the biggest number smaller than or equal to the specified value that is factorisable
      using only the <see cref="AcceptableDivisors"/>.
00034 /// </summary>
00035 /// <param name="n">The maximum number.</param>
00036 /// <returns>A number that is &lt;= <paramref name="n"/> and is factorisable using only the <see
      cref="AcceptableDivisors"/>.</returns>
00037         public static int GetAcceptableNumber(int n)
00038         {
00039             for (int i = n; i >= 1; i--)
00040             {
00041                 if (IsAcceptableNumber(i))
00042                 {
00043                     return i;
00044                 }
00045             }
00046
00047             throw new ArgumentOutOfRangeException(nameof(n), n, "The number must be strictly higher
      than 0!");
00048         }
00049
00050 /// <summary>
00051 /// Determine whether a number is factorisable using only the <see cref="AcceptableDivisors"/>.
00052 /// </summary>
00053 /// <param name="n">The number to analyse.</param>
00054 /// <returns>A boolean value indicating whether the number is factorisable using only the <see
      cref="AcceptableDivisors"/>.</returns>
00055         public static bool IsAcceptableNumber(int n)
00056         {
00057             if (n == 0)
00058             {
00059                 return false;
00060             }
00061             else if (n == 1)
00062             {
00063                 return true;
00064             }
00065             else
00066             {
00067                 for (int i = 0; i < AcceptableDivisors.Length; i++)
00068                 {
00069                     bool divided = false;
00070
00071                     while (n % AcceptableDivisors[i] == 0)
00072                     {
00073                         n /= AcceptableDivisors[i];
00074                         divided = true;
00075                     }
```

```
00076
00077                     if (divided)
00078                     {
00079                         return IsAcceptableNumber(n);
00080                     }
00081                 }
00082
00083                 return false;
00084             }
00085         }
00086
00087 /// <summary>
00088 /// Clear all pixels outside of a specified region.
00089 /// </summary>
00090 /// <param name="image">A pointer to the address where the pixel data is stored.</param>
00091 /// <param name="imageSize">The size in pixels of the image.</param>
00092 /// <param name="imageArea">The area represented by the image.</param>
00093 /// <param name="clipArea">The region outside which all pixels will be cleared, in image
      units.</param>
00094 /// <param name="pixelFormat">The format of the pixel data.</param>
00095         public static void ClipImage(IntPtr image, RoundedSize imageSize, Rectangle imageArea,
      Rectangle clipArea, PixelFormats pixelFormat)
00096         {
00097             int clipLeft = Math.Max(0, (int)Math.Ceiling((clipArea.X0 - imageArea.X0) /
      imageArea.Width * imageSize.Width - 0.001));
00098             int clipRight = Math.Max(0, (int)Math.Floor(imageSize.Width - (imageArea.X1 - clipArea.X1)
      / imageArea.Width * imageSize.Width + 0.001));
00099
00100             int clipTop = Math.Max(0, (int)Math.Ceiling((clipArea.Y0 - imageArea.Y0) /
      imageArea.Height * imageSize.Height - 0.001));
00101             int clipBottom = Math.Max(0, (int)Math.Floor(imageSize.Height - (imageArea.Y1 -
      clipArea.Y1) / imageArea.Height * imageSize.Height + 0.001));
00102
00103             int pixelSize = -1;
00104             byte clearValue = 0;
00105
00106             switch (pixelFormat)
00107             {
00108                 case PixelFormats.RGB:
00109                 case PixelFormats.BGR:
00110                     pixelSize = 3;
00111                     clearValue = 255;
00112                     break;
00113                 case PixelFormats.RGBA:
00114                 case PixelFormats.BGRA:
00115                     pixelSize = 4;
00116                     clearValue = 0;
00117                     break;
00118             }
00119
00120             int stride = imageSize.Width * pixelSize;
00121
00122             if (clipLeft > 0 || clipRight < imageSize.Width || clipTop > 0 || clipBottom <
      imageSize.Height)
00123             {
00124                 unsafe
00125                 {
00126                     byte* imageData = (byte*)image;
00127
00128                     for (int y = 0; y < imageSize.Height; y++)
00129                     {
00130                         if (y < clipTop || y >= clipBottom)
00131                         {
00132                             for (int x = 0; x < imageSize.Width; x++)
00133                             {
00134                                 for (int i = 0; i < pixelSize; i++)
00135                                 {
00136                                     imageData[y * stride + x * pixelSize + i] = clearValue;
00137                                 }
00138                             }
00139                         }
00140                         else
00141                         {
00142                             for (int x = 0; x < Math.Min(clipLeft, imageSize.Width); x++)
00143                             {
00144                                 for (int i = 0; i < pixelSize; i++)
00145                                 {
00146                                     imageData[y * stride + x * pixelSize + i] = clearValue;
00147                                 }
00148                             }
00149
00150                             for (int x = Math.Max(0, clipRight); x < imageSize.Width; x++)
00151                             {
00152                                 for (int i = 0; i < pixelSize; i++)
00153                                 {
00154                                     imageData[y * stride + x * pixelSize + i] = clearValue;
00155                                 }
```

```
00156                             }
00157                         }
00158                     }
00159                 }
00160             }
00161         }
00162
00163 /// <summary>
00164 /// Converts an image with premultiplied alpha values into an image with unpremultiplied alpha values.
00165 /// </summary>
00166 /// <param name="image">A pointer to the address where the pixel data is stored.</param>
00167 /// <param name="imageSize">The size in pixels of the image.</param>
00168         public static void UnpremultiplyAlpha(IntPtr image, RoundedSize imageSize)
00169         {
00170             int stride = imageSize.Width * 4;
00171
00172             unsafe
00173             {
00174                 byte* imageData = (byte*)image;
00175
00176                 for (int y = 0; y < imageSize.Height; y++)
00177                 {
00178                     for (int x = 0; x < imageSize.Width; x++)
00179                     {
00180                         if (imageData[y * stride + x * 4 + 3] > 0)
00181                         {
00182                             imageData[y * stride + x * 4] = (byte)(imageData[y * stride + x * 4] * 255
      / imageData[y * stride + x * 4 + 3]);
00183                             imageData[y * stride + x * 4 + 1] = (byte)(imageData[y * stride + x * 4 +
      1] * 255 / imageData[y * stride + x * 4 + 3]);
00184                             imageData[y * stride + x * 4 + 2] = (byte)(imageData[y * stride + x * 4 +
      2] * 255 / imageData[y * stride + x * 4 + 3]);
00185                         }
00186                     }
00187                 }
00188             }
00189         }
00190     }
00191 }
```

Index

Abort
MuPDFCore.MuPDFMultiThreadedPageRenderer, 59
AntiAliasing
MuPDFCore.MuPDFContext, 32
Avalonia, 21
Avalonia.Animation, 21
Avalonia.Animation.RectTransition, 119

Background
MuPDFCore.MuPDFRenderer.PDFRenderer, 106
BackgroundProperty
MuPDFCore.MuPDFRenderer.PDFRenderer, 102
Best
MuPDFCore.TesseractLanguage, 129
BestScripts
MuPDFCore.TesseractLanguage, 129
BlockIndex
MuPDFCore.MuPDFStructuredTextAddress, 71
BoundingBox
MuPDFCore.MuPDFStructuredTextBlock, 75
MuPDFCore.MuPDFStructuredTextLine, 80
BoundingQuad
MuPDFCore.MuPDFStructuredTextCharacter, 77
Bounds
MuPDFCore.MuPDFPage, 62

Character
MuPDFCore.MuPDFStructuredTextCharacter, 77
CharacterIndex
MuPDFCore.MuPDFStructuredTextAddress, 71
Characters
MuPDFCore.MuPDFStructuredTextLine, 80
ClearCache
MuPDFCore.MuPDFDocument, 39
ClearStore
MuPDFCore.MuPDFContext, 32
ClipToPageBounds
MuPDFCore.MuPDFDocument, 54
CodePoint
MuPDFCore.MuPDFStructuredTextCharacter, 77
Color
MuPDFCore.MuPDFStructuredTextCharacter, 77
CompareTo
MuPDFCore.MuPDFStructuredTextAddress, 67
Contain
MuPDFCore.MuPDFRenderer.PDFRenderer, 94
Contains
MuPDFCore.Quad, 111
MuPDFCore.Rectangle, 115
Count
MuPDFCore.MuPDFImageStructuredTextBlock, 57
MuPDFCore.MuPDFPageCollection, 64
MuPDFCore.MuPDFStructuredTextBlock, 75
MuPDFCore.MuPDFStructuredTextLine, 80
MuPDFCore.MuPDFStructuredTextPage, 85
MuPDFCore.MuPDFTextStructuredTextBlock, 88
Cover
MuPDFCore.MuPDFRenderer.PDFRenderer, 94
CreateDocument
MuPDFCore.MuPDFDocument, 39

Direction
MuPDFCore.MuPDFStructuredTextLine, 80
DisplayArea
MuPDFCore.MuPDFRenderer.PDFRenderer, 106
DisplayAreaProperty
MuPDFCore.MuPDFRenderer.PDFRenderer, 102
DisposableIntPtr
MuPDFCore.DisposableIntPtr, 26
Dispose
MuPDFCore.DisposableIntPtr, 26
MuPDFCore.MuPDFContext, 32
MuPDFCore.MuPDFDocument, 40
MuPDFCore.MuPDFMultiThreadedPageRenderer, 59
MuPDFCore.MuPDFPage, 62
MuPDFCore.MuPDFPageCollection, 64
DocumentOutputFileTypes
MuPDFCore, 23
DocumentRestrictions
MuPDFCore, 23

EncryptionState
MuPDFCore, 23
MuPDFCore.MuPDFDocument, 54
End
MuPDFCore.MuPDFStructuredTextAddressSpan, 72
Equals
MuPDFCore.MuPDFStructuredTextAddress, 67
ErrorCode
MuPDFCore.MuPDFException, 56
ExitCodes
MuPDFCore, 23
ExtractText
MuPDFCore.MuPDFDocument, 40, 41
ExtractTextAsync
MuPDFCore.MuPDFDocument, 41

Fast
MuPDFCore.TesseractLanguage, 129
FastScripts
MuPDFCore.TesseractLanguage, 129

GetClosestHitAddress
MuPDFCore.MuPDFStructuredTextPage, 83
GetEnumerator
MuPDFCore.MuPDFImageStructuredTextBlock, 57
MuPDFCore.MuPDFPageCollection, 64
MuPDFCore.MuPDFStructuredTextBlock, 74
MuPDFCore.MuPDFStructuredTextLine, 79
MuPDFCore.MuPDFStructuredTextPage, 83
MuPDFCore.MuPDFTextStructuredTextBlock, 87
GetHashCode
MuPDFCore.MuPDFStructuredTextAddress, 68
GetHighlightQuads
MuPDFCore.MuPDFStructuredTextPage, 83
GetHitAddress
MuPDFCore.MuPDFStructuredTextPage, 84
GetMultiThreadedRenderer
MuPDFCore.MuPDFDocument, 42
GetProgress
MuPDFCore.MuPDFMultiThreadedPageRenderer, 59
MuPDFCore.MuPDFRenderer.PDFRenderer, 94
GetRenderedSize
MuPDFCore.MuPDFDocument, 42, 43
GetSelectedText
MuPDFCore.MuPDFRenderer.PDFRenderer, 94
GetSpanItem
MuPDFCore.MuPDFMultiThreadedPageRenderer, 59
GetStructuredTextPage
MuPDFCore.MuPDFDocument, 43, 44
GetStructuredTextPageAsync
MuPDFCore.MuPDFDocument, 44
GetText
MuPDFCore.MuPDFStructuredTextPage, 84
GraphicsAntiAliasing
MuPDFCore.MuPDFContext, 32

Height
MuPDFCore.Rectangle, 118
MuPDFCore.RoundedRectangle, 123
MuPDFCore.RoundedSize, 125
MuPDFCore.Size, 127
HighlightBrush
MuPDFCore.MuPDFRenderer.PDFRenderer, 106
HighlightBrushProperty
MuPDFCore.MuPDFRenderer.PDFRenderer, 102
HighlightedRegions
MuPDFCore.MuPDFRenderer.PDFRenderer, 106
HighlightedRegionsProperty
MuPDFCore.MuPDFRenderer.PDFRenderer, 103

Increment
MuPDFCore.MuPDFStructuredTextAddress, 68
Initialize
MuPDFCore.MuPDFRenderer.PDFRenderer, 95–97
InitializeAsync
MuPDFCore.MuPDFRenderer.PDFRenderer, 97–100
InputFileTypes
MuPDFCore, 23
Intersect
MuPDFCore.Rectangle, 116
IsViewerInitialized
MuPDFCore.MuPDFRenderer.PDFRenderer, 107
IsViewerInitializedProperty
MuPDFCore.MuPDFRenderer.PDFRenderer, 103

Language
MuPDFCore.TesseractLanguage, 132
Layout
MuPDFCore.MuPDFDocument, 45
LayoutSinglePage
MuPDFCore.MuPDFDocument, 45
Length
MuPDFCore.MuPDFPageCollection, 64
LineIndex
MuPDFCore.MuPDFStructuredTextAddress, 71
Lines
MuPDFCore.MuPDFTextStructuredTextBlock, 88
LowerLeft
MuPDFCore.Quad, 112
LowerRight
MuPDFCore.Quad, 112

MaxProgress
MuPDFCore.RenderProgress.ThreadRenderProgress, 133
Message
MuPDFCore.MessageEventArgs, 28
MessageEventArgs
MuPDFCore.MessageEventArgs, 28
MuPDFContext
MuPDFCore.MuPDFContext, 31
MuPDFCore, 21
DocumentOutputFileTypes, 23
DocumentRestrictions, 23
EncryptionState, 23
ExitCodes, 23
InputFileTypes, 23
PasswordTypes, 24
PixelFormats, 24
RasterOutputFileTypes, 24
RestrictionState, 24
MuPDFCore.DisposableIntPtr, 25
DisposableIntPtr, 26
Dispose, 26
MuPDFCore.DocumentLockedException, 27
MuPDFCore.MessageEventArgs, 27
Message, 28
MessageEventArgs, 28
MuPDFCore.MuPDF, 29
RedirectOutput, 29

ResetOutput, 29
StandardErrorMessage, 30
StandardOutputMessage, 30
MuPDFCore.MuPDFContext, 30
AntiAliasing, 32
ClearStore, 32
Dispose, 32
GraphicsAntiAliasing, 32
MuPDFContext, 31
ShrinkStore, 32
StoreMaxSize, 33
StoreSize, 33
TextAntiAliasing, 33
MuPDFCore.MuPDFDocument, 34
ClearCache, 39
ClipToPageBounds, 54
CreateDocument, 39
Dispose, 40
EncryptionState, 54
ExtractText, 40, 41
ExtractTextAsync, 41
GetMultiThreadedRenderer, 42
GetRenderedSize, 42, 43
GetStructuredTextPage, 43, 44
GetStructuredTextPageAsync, 44
Layout, 45
LayoutSinglePage, 45
MuPDFDocument, 37, 38
Pages, 54
Render, 46–48
Restrictions, 55
RestrictionState, 55
SaveImage, 49
SaveImageAsJPEG, 50, 51
TryUnlock, 51
WriteImage, 52, 53
WriteImageAsJPEG, 53, 54
MuPDFCore.MuPDFException, 55
ErrorCode, 56
MuPDFCore.MuPDFImageStructuredTextBlock, 56
Count, 57
GetEnumerator, 57
this[int index], 57
Type, 58
MuPDFCore.MuPDFMultiThreadedPageRenderer, 58
Abort, 59
Dispose, 59
GetProgress, 59
GetSpanItem, 59
Render, 60
ThreadCount, 61
MuPDFCore.MuPDFPage, 61
Bounds, 62
Dispose, 62
PageNumber, 62
MuPDFCore.MuPDFPageCollection, 63
Count, 64
Dispose, 64
GetEnumerator, 64
Length, 64
this[int index], 64
MuPDFCore.MuPDFRenderer, 24
MuPDFCore.MuPDFRenderer.PDFRenderer, 90
Background, 106
BackgroundProperty, 102
Contain, 94
Cover, 94
DisplayArea, 106
DisplayAreaProperty, 102
GetProgress, 94
GetSelectedText, 94
HighlightBrush, 106
HighlightBrushProperty, 102
HighlightedRegions, 106
HighlightedRegionsProperty, 103
Initialize, 95–97
InitializeAsync, 97–100
IsViewerInitialized, 107
IsViewerInitializedProperty, 103
PageBackground, 107
PageBackgroundProperty, 103
PageNumber, 107
PageNumberProperty, 103
PageSize, 107
PageSizeProperty, 104
PDFRenderer, 93
PointerEventHandlers, 93
PointerEventHandlersType, 107
PointerEventHandlerTypeProperty, 104
ReleaseResources, 100
Render, 100
RenderThreadCount, 108
RenderThreadCountProperty, 104
Search, 101
SelectAll, 101
Selection, 108
SelectionBrush, 108
SelectionBrushProperty, 104
SelectionProperty, 105
SetDisplayAreaNow, 101
Zoom, 108
ZoomEnabled, 108
ZoomEnabledProperty, 105
ZoomIncrement, 109
ZoomIncrementProperty, 105
ZoomProperty, 105
ZoomStep, 102
MuPDFCore.MuPDFRenderer/PDFRenderer.cs, 135
MuPDFCore.MuPDFRenderer/PDFRenderer.Properties.cs, 155
MuPDFCore.MuPDFRenderer/RectTransition.cs, 159
MuPDFCore.MuPDFStructuredTextAddress, 65
BlockIndex, 71
CharacterIndex, 71
CompareTo, 67
Equals, 67

GetHashCode, 68
Increment, 68
LineIndex, 71
MuPDFStructuredTextAddress, 66
operator!=, 68
operator<, 69
operator<=, 69
operator>, 70
operator>=, 70
operator==, 69
MuPDFCore.MuPDFStructuredTextAddressSpan, 72
End, 72
MuPDFStructuredTextAddressSpan, 72
Start, 73
MuPDFCore.MuPDFStructuredTextBlock, 73
BoundingBox, 75
Count, 75
GetEnumerator, 74
this[int index], 75
Type, 75
Types, 74
MuPDFCore.MuPDFStructuredTextCharacter, 76
BoundingQuad, 77
Character, 77
CodePoint, 77
Color, 77
Origin, 77
Size, 77
ToString, 76
MuPDFCore.MuPDFStructuredTextLine, 78
BoundingBox, 80
Characters, 80
Count, 80
Direction, 80
GetEnumerator, 79
Text, 81
this[int index], 81
ToString, 79
WritingMode, 81
WritingModes, 79
MuPDFCore.MuPDFStructuredTextPage, 82
Count, 85
GetClosestHitAddress, 83
GetEnumerator, 83
GetHighlightQuads, 83
GetHitAddress, 84
GetText, 84
Search, 85
StructuredTextBlocks, 85
this[int index], 85
this[MuPDFStructuredTextAddress address], 86
MuPDFCore.MuPDFTextStructuredTextBlock, 86
Count, 88
GetEnumerator, 87
Lines, 88
this[int index], 88
ToString, 88
Type, 89
MuPDFCore.OCRProgressInfo, 89
Progress, 89
MuPDFCore.PointF, 109
PointF, 109
X, 110
Y, 110
MuPDFCore.Quad, 110
Contains, 111
LowerLeft, 112
LowerRight, 112
Quad, 111
UpperLeft, 112
UpperRight, 112
MuPDFCore.Rectangle, 113
Contains, 115
Height, 118
Intersect, 116
Rectangle, 114
Round, 116
Split, 117
ToQuad, 117
Width, 118
X0, 117
X1, 118
Y0, 118
Y1, 118
MuPDFCore.RenderProgress, 119
ThreadRenderProgresses, 120
MuPDFCore.RenderProgress.ThreadRenderProgress, 133
MaxProgress, 133
Progress, 133
MuPDFCore.RoundedRectangle, 120
Height, 123
RoundedRectangle, 121
Split, 122
Width, 123
X0, 122
X1, 122
Y0, 122
Y1, 123
MuPDFCore.RoundedSize, 123
Height, 125
RoundedSize, 124
Split, 124
Width, 125
MuPDFCore.Size, 125
Height, 127
Size, 126
Split, 127
Width, 127
MuPDFCore.TesseractLanguage, 128
Best, 129
BestScripts, 129
Fast, 129
FastScripts, 129
Language, 132
Prefix, 132

TesseractLanguage, 130–132
MuPDFCore/MuPDF.cs, 160
MuPDFCore/MuPDFContext.cs, 179
MuPDFCore/MuPDFDisplayList.cs, 181
MuPDFCore/MuPDFDocument.cs, 183
MuPDFCore/MuPDFMultiThreadedPageRenderer.cs, 206
MuPDFCore/MuPDFPage.cs, 214
MuPDFCore/MuPDFStructuredTextPage.cs, 216
MuPDFCore/Rectangles.cs, 233
MuPDFCore/TesseractLanguage.cs, 241
MuPDFCore/Utils.cs, 261
MuPDFDocument
MuPDFCore.MuPDFDocument, 37, 38
MuPDFStructuredTextAddress
MuPDFCore.MuPDFStructuredTextAddress, 66
MuPDFStructuredTextAddressSpan
MuPDFCore.MuPDFStructuredTextAddressSpan, 72

operator!=
MuPDFCore.MuPDFStructuredTextAddress, 68
operator<
MuPDFCore.MuPDFStructuredTextAddress, 69
operator<=
MuPDFCore.MuPDFStructuredTextAddress, 69
operator>
MuPDFCore.MuPDFStructuredTextAddress, 70
operator>=
MuPDFCore.MuPDFStructuredTextAddress, 70
operator==
MuPDFCore.MuPDFStructuredTextAddress, 69
Origin
MuPDFCore.MuPDFStructuredTextCharacter, 77

PageBackground
MuPDFCore.MuPDFRenderer.PDFRenderer, 107
PageBackgroundProperty
MuPDFCore.MuPDFRenderer.PDFRenderer, 103
PageNumber
MuPDFCore.MuPDFPage, 62
MuPDFCore.MuPDFRenderer.PDFRenderer, 107
PageNumberProperty
MuPDFCore.MuPDFRenderer.PDFRenderer, 103
Pages
MuPDFCore.MuPDFDocument, 54
PageSize
MuPDFCore.MuPDFRenderer.PDFRenderer, 107
PageSizeProperty
MuPDFCore.MuPDFRenderer.PDFRenderer, 104
PasswordTypes
MuPDFCore, 24
PDFRenderer
MuPDFCore.MuPDFRenderer.PDFRenderer, 93
PixelFormats
MuPDFCore, 24
PointerEventHandlers
MuPDFCore.MuPDFRenderer.PDFRenderer, 93
PointerEventHandlersType
MuPDFCore.MuPDFRenderer.PDFRenderer, 107
PointerEventHandlerTypeProperty
MuPDFCore.MuPDFRenderer.PDFRenderer, 104
PointF
MuPDFCore.PointF, 109
Prefix
MuPDFCore.TesseractLanguage, 132
Progress
MuPDFCore.OCRProgressInfo, 89
MuPDFCore.RenderProgress.ThreadRenderProgress, 133

Quad
MuPDFCore.Quad, 111

RasterOutputFileTypes
MuPDFCore, 24
Rectangle
MuPDFCore.Rectangle, 114
RedirectOutput
MuPDFCore.MuPDF, 29
ReleaseResources
MuPDFCore.MuPDFRenderer.PDFRenderer, 100
Render
MuPDFCore.MuPDFDocument, 46–48
MuPDFCore.MuPDFMultiThreadedPageRenderer, 60
MuPDFCore.MuPDFRenderer.PDFRenderer, 100
RenderThreadCount
MuPDFCore.MuPDFRenderer.PDFRenderer, 108
RenderThreadCountProperty
MuPDFCore.MuPDFRenderer.PDFRenderer, 104
ResetOutput
MuPDFCore.MuPDF, 29
Restrictions
MuPDFCore.MuPDFDocument, 55
RestrictionState
MuPDFCore, 24
MuPDFCore.MuPDFDocument, 55
Round
MuPDFCore.Rectangle, 116
RoundedRectangle
MuPDFCore.RoundedRectangle, 121
RoundedSize
MuPDFCore.RoundedSize, 124

SaveImage
MuPDFCore.MuPDFDocument, 49
SaveImageAsJPEG
MuPDFCore.MuPDFDocument, 50, 51
Search
MuPDFCore.MuPDFRenderer.PDFRenderer, 101
MuPDFCore.MuPDFStructuredTextPage, 85
SelectAll
MuPDFCore.MuPDFRenderer.PDFRenderer, 101
Selection
MuPDFCore.MuPDFRenderer.PDFRenderer, 108
SelectionBrush
MuPDFCore.MuPDFRenderer.PDFRenderer, 108

SelectionBrushProperty
MuPDFCore.MuPDFRenderer.PDFRenderer, 104
SelectionProperty
MuPDFCore.MuPDFRenderer.PDFRenderer, 105
SetDisplayAreaNow
MuPDFCore.MuPDFRenderer.PDFRenderer, 101
ShrinkStore
MuPDFCore.MuPDFContext, 32
Size
MuPDFCore.MuPDFStructuredTextCharacter, 77
MuPDFCore.Size, 126
Split
MuPDFCore.Rectangle, 117
MuPDFCore.RoundedRectangle, 122
MuPDFCore.RoundedSize, 124
MuPDFCore.Size, 127
StandardErrorMessage
MuPDFCore.MuPDF, 30
StandardOutputMessage
MuPDFCore.MuPDF, 30
Start
MuPDFCore.MuPDFStructuredTextAddressSpan, 73
StoreMaxSize
MuPDFCore.MuPDFContext, 33
StoreSize
MuPDFCore.MuPDFContext, 33
StructuredTextBlocks
MuPDFCore.MuPDFStructuredTextPage, 85

TesseractLanguage
MuPDFCore.TesseractLanguage, 130–132
Text
MuPDFCore.MuPDFStructuredTextLine, 81
TextAntiAliasing
MuPDFCore.MuPDFContext, 33
this[int index]
MuPDFCore.MuPDFImageStructuredTextBlock, 57
MuPDFCore.MuPDFPageCollection, 64
MuPDFCore.MuPDFStructuredTextBlock, 75
MuPDFCore.MuPDFStructuredTextLine, 81
MuPDFCore.MuPDFStructuredTextPage, 85
MuPDFCore.MuPDFTextStructuredTextBlock, 88
this[MuPDFStructuredTextAddress address]
MuPDFCore.MuPDFStructuredTextPage, 86
ThreadCount
MuPDFCore.MuPDFMultiThreadedPageRenderer, 61
ThreadRenderProgresses
MuPDFCore.RenderProgress, 120
ToQuad
MuPDFCore.Rectangle, 117
ToString
MuPDFCore.MuPDFStructuredTextCharacter, 76
MuPDFCore.MuPDFStructuredTextLine, 79
MuPDFCore.MuPDFTextStructuredTextBlock, 88
TryUnlock
MuPDFCore.MuPDFDocument, 51
Type
MuPDFCore.MuPDFImageStructuredTextBlock, 58
MuPDFCore.MuPDFStructuredTextBlock, 75
MuPDFCore.MuPDFTextStructuredTextBlock, 89
Types
MuPDFCore.MuPDFStructuredTextBlock, 74

UpperLeft
MuPDFCore.Quad, 112
UpperRight
MuPDFCore.Quad, 112

Width
MuPDFCore.Rectangle, 118
MuPDFCore.RoundedRectangle, 123
MuPDFCore.RoundedSize, 125
MuPDFCore.Size, 127
WriteImage
MuPDFCore.MuPDFDocument, 52, 53
WriteImageAsJPEG
MuPDFCore.MuPDFDocument, 53, 54
WritingMode
MuPDFCore.MuPDFStructuredTextLine, 81
WritingModes
MuPDFCore.MuPDFStructuredTextLine, 79

X
MuPDFCore.PointF, 110
X0
MuPDFCore.Rectangle, 117
MuPDFCore.RoundedRectangle, 122
X1
MuPDFCore.Rectangle, 118
MuPDFCore.RoundedRectangle, 122

Y
MuPDFCore.PointF, 110
Y0
MuPDFCore.Rectangle, 118
MuPDFCore.RoundedRectangle, 122
Y1
MuPDFCore.Rectangle, 118
MuPDFCore.RoundedRectangle, 123

Zoom
MuPDFCore.MuPDFRenderer.PDFRenderer, 108
ZoomEnabled
MuPDFCore.MuPDFRenderer.PDFRenderer, 108
ZoomEnabledProperty
MuPDFCore.MuPDFRenderer.PDFRenderer, 105
ZoomIncrement
MuPDFCore.MuPDFRenderer.PDFRenderer, 109
ZoomIncrementProperty
MuPDFCore.MuPDFRenderer.PDFRenderer, 105
ZoomProperty
MuPDFCore.MuPDFRenderer.PDFRenderer, 105
ZoomStep
MuPDFCore.MuPDFRenderer.PDFRenderer, 102